

# Web Application Modules (WAMs)

[Before You Begin](#)

[An Introduction to WAMs](#)

[WAMs Deconstructed](#)

[Essential Topics](#)

[Advanced Topics](#)

[Execute WAM Applications](#)

[WAM and WEBEVENT Interoperability](#)

[Technology Services](#)

[Weblets for XHTML Technology Service](#)

[Weblets for jQuery Mobile Technology Service](#)

[WAM Tutorials](#)

[Appendix A. XSL and XML Conformance](#)

[Appendix B. WAM XML Structure](#)

[Appendix C. Deprecated Weblets](#)

[WAM Wizards](#)

Edition Date May 29, 2015

© LANSa

## Before You Begin

- The LANSAs for the Web WAM technology is an extension of the LANSAs development environment. If you have Repository and RDML/RDMLX skills, you can apply them to building Web-based applications.
- Before any development efforts begin, you must have a properly installed and configured LANSAs for the Web system that supports the WAM technology.
- You will need a properly configured Web Server and a properly configured LANSAs Application/Data Server.
- Your development system must have Internet Explorer 6.0 or later and MSXML Core Services 6.0 XML Parser. The MSXML parser is installed during the Visual LANSAs install.
- For details about the installation and configuration of LANSAs for the Web, refer to the *Installing LANSAs on Windows Guide*.
- If you are using WAMs for the very first time, we recommended that you read [An Introduction to WAMs](#) which will give you an outline of all the components of WAMs and [WAMs Deconstructed](#), which will give you a more in-depth understanding of the WAM components. We also recommend that you complete the WAM Tutorials. For information about the WAM tutorials refer to [WAM Tutorials](#).
- Weblets are shipped with the software to provide wizards or building blocks that facilitate the rapid development of HTML browser-based applications. To create your own Weblets, or complex HTML pages, you will need XSL, HTML and JavaScript skills.
- The XSL used for transformation of the XML WEBROUTINE document conforms to the standard W3C XSL 1.0 specification. Refer to XSL 1.0 references for information.

## **1. An Introduction to WAMs**

This chapter gives you a very simple overview of LANSA's WAM architecture and explains each component.

This introduction doesn't describe the minutiae of every technology, command and parameter. Rather, it is a solid overview so that you can approach the use of WAMs with confidence. It is assumed that you have a working knowledge of the Visual LANSA Application Development Environment.

### **Why did LANSA develop WAM technology?**

There are a number of reasons, but two key points stand out:

1. Web technologies are evolving very quickly. HTML is no longer the only way to deliver web content. It is imperative that web application development in LANSA is readily adaptable to new technologies as they emerge.
2. It is clear that application development is heading towards a component-based future so it is imperative that web application development in LANSA allows you to take full advantage of component-based techniques.

## 1.1 What is a WAM?

Web Application Modules (WAMs) are LANSAs solution for building applications that deliver their User Interface in a form of XML, typically via a web browser.

WAMs consist of two distinct pieces: an Application Logic Layer and a Presentation Layer.

Here's what happens:

The **Application Logic Layer** is provided by RDMLX code, maintained in the LANSAs Editor. The data output by the Application Logic Layer and destined for the Presentation Layer is in the form of XML.

the **Presentation Layer** is also maintained in the LANSAs Editor. When you create your Application Logic Layer you generate a default "best guess" Presentation User Interface in XSL. The RDMLX does not need to be compiled to generate the Presentation User Interface. Once the XSL is generated you can update it using the LANSAs Editor.

Your business data (in the form of XML) is exchanged between these two layers by means of familiar fields and working lists. The following diagram summarizes the architecture:



## **1.2 The BIG Advantage of the WAM Architecture**

The separation of Application Logic and Presentation Layers introduces a new level of flexibility to LANSA's web solution.

Separating the business logic from the User Interface helps to "future-proof" your WAM applications. Potentially, your XHTML (eXtensible Hypertext Markup Language) browser-based applications of today can be deployed tomorrow using the User Interface technology of the day (whether that be XAML, AUIML or some other technology yet to be invented) without having to change your business logic.

the Presentation Layer uses a Technology Service (TSP) to generate the User Interface for each potential platform. Currently, one of the most common Technology Services in the web world is XHTML. This is one TSP that LANSA generates for, delivering a User Interface that will look good in Internet Explorer or some other browser.

The other TSP that LANSA generates for is PocketPC XHTML, which is basically the same as the default XHTML but designed to fit a handheld device.

Other TSPs can be introduced with relative ease, allowing a single WAM to have multiple user interfaces by selecting the appropriate TSP. This means that the same application can be run on different devices, making for the perfect separation of Application Logic and presentation.

## 1.3 Other Great Things about WAMs

There are other powerful advantages to adopting LANSA's WAM technology. In the Presentation Layer these include:

- **Industry standard architecture:** WAMs are based on industry standard technologies including XML and XSL. This ensures that your WAM applications are open and flexible.
- **An editor:** the LANSA Editor lets you "paint" your WAM's User Interface using point-and-click. When more power is needed, the UI can be edited at source code level with round-trip support, i.e. you can edit the source code and then revert to using the LANSA Editor for further work.
- **Shipped and user-definable "Weblets":** Weblets are XSL-based components used for encapsulating common field visualizations and other User Interface elements. Weblets are designed to promote re-use in the Presentation Layer. LANSA ships with a range of ready-to-use Weblets for common User Interface elements, but you can also build your own.

In the Application Logic Layer, WAMs exploit and build on much of LANSA's traditional application development strengths:

- **Repository-based:** LANSA's repository-based approach to application development captures business rules and domain knowledge and ensures that it is consistently applied throughout an application.
- **Component-based:** in LANSA parlance, WAMs are components and are capable of making use of other LANSA components. This offers the potential for you to build upon the repository-based approach and further extend the separation between business rules, Application Logic and the User Interface. You can build common business logic components that can be shared between browser-based applications, rich-client applications and integration projects.
- **Single skill set:** the Application Logic for WAMs is built using the same RDML programming language that is used throughout LANSA. For example, LANSA developers with 5250 green screen backgrounds can quickly and easily learn how to produce sophisticated web browser-based applications.

Let's now look more closely at the key components of the Application Logic and Presentation Layers.

## **1.4 The Application Logic Layer**

WAMs, as the name indicates, are the modules of a web application and contain your application's logic. The Application Logic Layer contains RDMLX code, maintained in the LANSAs Editor.

You can have as many or as few WAMs making up your application as you like. An entire application can be contained in a single WAM, or it can be spread across multiple WAMs.

## 1.4.1 Webroutines

A WAM may contain one or more Webroutines. These contain your Application Logic. They can be thought of in much the same way as subroutines. Indeed, a Webroutine is defined in a similar manner to a subroutine, using the Webroutine/Endroutine command combination, as follows:

```
Webroutine Name(MyWebroutine)
|
| * your RDMLX code here.
|
Endroutine
```

To execute this Webroutine from a browser (i.e. to initiate this Webroutine), a user would enter a URL that looks something like this:

**[http://www.MyWebSite.com/cgi-bin/lansaweb?  
wam=MYWAM&webrtn=MyWebroutine](http://www.MyWebSite.com/cgi-bin/lansaweb?wam=MYWAM&webrtn=MyWebroutine)**

Note that, when the Endroutine statement is reached, control is passed back to the Presentation Layer.

What the Presentation Layer shows is the web page associated with that Webroutine. Yes, each Webroutine is capable of having its own web page that it shows to the user. More about this in [1.5 The Presentation Layer](#).

Of course, you will often want to pass data back and forth between the Application Logic Layer and the Presentation Layer. This is done with [1.4.2 Web Maps](#).

## 1.4.2 Web Maps

Web Maps are the interfaces for the exchange of data between your Webroutines and the Presentation Layer.

The specification of a web map that outputs a Department description to the Presentation Layer would look something like this:

```
Webroutine Name(MyWebroutine)
  Web_Map For(*output) Fields(#DEPTDESC)
  * your RDMLX code here.
Endroutine
```

Note the *For* parameter of the Web\_Map command.

The parameter values can be:

- \*input* – this defines data that is coming from the Presentation Layer into the Webroutine.

- \*output* – this defines data that is going from the Webroutine to the Presentation Layer.

- \*both* – this defines data that comes into the Webroutine from the Presentation Layer and is sent from the Webroutine to the Presentation Layer.

The fourth value, *\*none* is described in [1.4.3 Being Stateless](#).

Say you had a Webroutine whose job it was to accept a Department Code as input and then to output a Department Description. The specification of its Web Maps would look like this:

```
Webroutine Name(MyWebroutine)
  Web_Map For(*input) Fields(#DEPARTMENT)
  Web_Map For(*output) Fields(#DEPTDESC)
  * some code to fetch the department description from a file here.
Endroutine
```

The department code is *\*input* to the Webroutine when it is executed and the Department Description is *\*output* to the Presentation Layer when the Webroutine ends. These settings are very strict. That is, the value of DEPARTMENT, as specified in this example, is only ever input to the Webroutine. Once the Webroutine ends, its value is not sent back to the Presentation Layer. Similarly, when the Webroutine begins executing, DEPTDESC has no value, as it is only ever output by the Webroutine.

This is where the *For* parameter value of *\*both* is very useful if your Webroutine needs to accept certain information as input as well as to be able to output it. A classic example of this is that of a simple file maintenance application. When a user wants to edit information about a Department, , the Webroutine should *\*output* that information to the Presentation Layer in order to show it, but it should also be able to accept it as *\*input*, as it may have been changed by the user. So the web map might look a bit like this:

```
Webroutine Name(MyWebroutine)
  Web_Map For(*both) Fields(#DEPARTMENT #DEPTDESC)
  * some code to fetch the department description from a file here.
Endroutine
```

As well as fields, working lists can be used to pass data back and forth. Simply specify the name of the working list on your Web Maps, just like this:

You can either use the identifier or the name of the field in the Web\_Map command. The XSL and XML will be generated using the identifier. The WAM Editor will in most cases display the name but still use the identifier under the covers.

```
Def_List Name(#WLDEPTS) Fields(#DEPARTMENT #DEPTDESC) Type(*Working)

Webroutine Name(MyWebroutine)
  Web_Map For(*both) Fields(#WLDEPTS)
  * some code to fetch the department description from a file here.
Endroutine
```

## Attributes for Fields in Web Maps

Note the *For* parameter of the `Web_Map` command affects which fields and working lists are exchanged between the Presentation Layer and the Webroutine. You might find it helpful to think of all the relevant `Web_Map` commands as defining a parameter list for your Webroutine.

However, the *For* parameter of the `Web_Map` command does not affect how the field is presented – that is the job of the Presentation Layer.

While you may want certain values to be for (*\*both*), you may not want the user to be able to change them, or indeed see them, on the web page. To do this, you can specify the following attributes for fields in Web Maps:

*\*output* – the data is displayed on the web page, but the user cannot change it.

*\*hidden* – the data is passed between the logic and Presentation Layers, but the user cannot see it.

*\*private* – the data is passed between the logic and Presentation Layers, but the data is not merged into the XSL. (This has specialized uses where the value of a field or contents of a list are referenced during the XSL transformation process but do not need to be generated into the final page).

Note: *\*input* is also a valid attribute, and is the default.

Remember, though, that how the field is represented is the responsibility of the Presentation Layer. The field attributes you set in your web map to control this are only effective when the web design is generated for the Webroutine. After that you can alter how the field is represented using the LANSAs Editor. What this also means is that *field attributes specified in the web map should only be regarded as an indication of your intentions*, as ultimately the Presentation Layer controls how the field is represented.

Here's an example where a DEPARTMENT is to be passed back and forth and displayed, but you don't want the user changing its value:

```
Webroutine Name(MyWebroutine)
  Web_Map For(*both) Fields((#DEPARTMENT *output) #DEPTDESC)
  * some code to fetch the department description from a file here.
Endroutine
```

Particular things to note about this example are:

- The only effect of specifying the *\*output* field attribute is to make field

#DEPARTMENT output-only when generating the web design for the Webroutine – it does not mean that the field remains as output-only on the web page as it may be changed using the LANSA Editor.

- If field #DEPARTMENT is output-only on the web page, the web page will not POST the field value back to the next Webroutine. For example, if the Webroutine posts its data back to the same Webroutine, then it will not work because the value of field #DEPARTMENT, being output-only, will not be posted. If, however, this Webroutine is invoked by some other Webroutine that does POST a value for field #DEPARTMENT, then this use will work.

As you can see, specifying *\*output* as a field attribute has a very different meaning and effect to using *for(\*output)* on the Web\_Map command.

## **Webroutines generate Presentation Layer XSL when Compiled**

When a Webroutine that has Web Maps is compiled, appropriate XSL for the Presentation Layer is generated. In the case of *\*input* and *\*both* Web Maps, LANSA will typically generate a web page for the Webroutine that displays those fields and/or working lists.

Exactly what is generated for the Presentation Layer is described in [1.5.2 What is Generated for the Presentation Layer by Default?](#)

## Global Web Maps

If you specify a web map outside the bounds of a Webroutine, it becomes a global web map. This means that every Webroutine in the WAM adopts that web map.

You will find many uses for global Web Maps, and you will see examples of using them in this guide, such as in [1.4.3 Being Stateless](#).

Until now, we have described Web Maps that have been specified in Webroutines such as in this example:

```
Begin_Com Role(*EXTENDS #PRIM_WAM)
  Def_List Name(#WLDEPTS) Fields(#DEPARTMENT #DEPTDESC) Type(*Working)
  Webroutine Name(MyWebroutine)
    Web_Map For(*both) Fields(#SECTION #SECDESC)
    Web_Map For(*both) Fields(#WLDEPTS)
    * your RDMLX code here.
  Endroutine
  Webroutine Name(MyOtherWebroutine)
    Web_Map For(*both) Fields(#WLDEPTS)
    * some more RDMLX code here.
  Endroutine
End_Com
```

In the following global web map example, note the placement of the web map for the WLDEPTS working list. Both *MyWebroutine* and *MyOtherWebroutine* will adopt this web map when compiled. Note also that *MyWebroutine* also has its own, local, web map specified.

```
Begin_Com Role(*EXTENDS #PRIM_WAM)

  Def_List Name(#WLDEPTS) Fields(#DEPARTMENT #DEPTDESC) Type(*Working)

  Web_Map For(*both) Fields(#WLDEPTS)

  Webroutine Name(MyWebroutine)
    Web_Map For(*both) Fields(#SECTION #SECDESC)

    * your RDMLX code here.
  Endroutine

  Webroutine Name(MyOtherWebroutine)

    * some more RDMLX code here.
  Endroutine

End_Com
```

## Controlling Webroutine Flow Programmatically

As well as Webroutines being executed from the Presentation Layer, which is described later, they can also be executed under program control using the Call and Transfer RDML commands. Consider the following code:

```
Begin_Com Role(*EXTENDS #PRIM_WAM)
  Webroutine Name(Initialize)
    Web_Map For(*output) Fields(#DEPARTMENT)

    * some code to derive a value for DEPARTMENT here.

    Transfer To routine(ShowPage)
  Endroutine

  Webroutine Name(ShowPage)
    Web_Map For(*both) Fields(#DEPARTMENT)
  Endroutine
End_Com
```

Imagine that the user executes the *Initialize* Webroutine by keying a URL into their browser. It places a value into DEPARTMENT.

The Transfer command then transfers control to the *ShowPage* Webroutine, which accepts DEPARTMENT's value. Note that the appropriate mapping takes place as part of the transfer, according to the web map definitions for the current Webroutine. The *ShowPage* Webroutine doesn't actually do anything, but ends, outputting DEPARTMENT's value as it does. Control is then passed to the Presentation Layer, which shows *ShowPage's* web page.

Note that the Endroutine command of the *Initialize* Webroutine is never executed, as control is transferred to ShowPage. Consequently, no page shows for the *Initialize* Webroutine.

This use of Transfer illustrates a slightly more advanced technique, whereby a single Webroutine in the WAM can be responsible for showing the Presentation Layer. Using this technique, all other Webroutines can be used solely to perform Application Logic. All of them transfer control to the *ShowPage* Webroutine as the last thing they do.

You may find that this technique can simplify your application. What this means is that your application may be made up of multiple WAMs, each WAM using only one Webroutine to display a web page. To further support this, the Transfer command also supports transfer between WAMs:

Transfer Toroutine(#MYWAM99.ShowPage)

The Call command is similar to the Transfer command in that it executes a Webroutine. The difference is that control is not transferred – the Webroutine is executed fully, right through to the Endroutine command and control is then returned to the executing Webroutine. Note that the web page for the called Webroutine is not shown.

### 1.4.3 Being Stateless

One of the key points about WAMs is that they are stateless. In fact, any internet-based application is stateless. What this means is that when a WAM is executed from the Presentation Layer, it runs (a job is initiated on the server), produces some output (a web page), and then ends (the job on the server ends and control is transferred back to the browser).

The job starting and ending, to all intents and purposes, is a "transaction". Any data that needs to be maintained for the user's web "session", i.e. span multiple transactions, must be kept somewhere. A good example of this is when a user logs on to a website. The fact that they're logged on needs to be maintained across multiple transactions, as certain functions of the application may only be executed if they are logged on.

In a windows or green screen application, this is all implicit – the user's job stays alive in memory, waiting for the next move from the user. But on the web, we don't know what the user might do next – they might key in a new URL, hit the back button, close the browser and so on.

Data to be displayed on the web page is moved back and forth courtesy of Web Maps. Even data that is hidden is moved back and forth. But this shouldn't happen in the case of "session" data. This is data that is important to the Application Logic Layer only, and doesn't influence what happens in the Presentation Layer.

WAMs supports this idea of session data by using a special value of *\*none* in the *For* parameter of the web map and a value of *\*persist* in the web map's options parameter. A web map to maintain the log on status of a user might look like this:

```
Begin_Com Role(*EXTENDS #PRIM_WAM)
  Web_Map For(*none) Fields(#LOGGEDON) Options(*PERSIST)
  Webroutine Name(ShowPage)
    Web_Map For(*both) Fields(#DEPARTMENT)
  Endroutine
End_Com
```

Note the placement of the web map. Because it is global, all Webroutines in the WAM will adopt it. It makes a lot of sense to code your session data Web Maps in this way – so that the data is mapped in and out at all times.

There are other parameters and settings that need to be taken into account when coding for session data, but you should now have the basic idea. Refer to [WAM Session Management](#) for more information.

## 1.4.4 How Reusable Parts can Play a Role

As with all applications, internet-based or not, there may be functionality that needs to be used in more than one place. Again, a good example is the "log on" function (recording the user id, date and time of logon in a database, etc), which may be the same across different WAM applications.

LANSA's Reusable Parts can provide the necessary level of modularity. Simply use the Define\_com command to declare the appropriate parts and use them for the Application Logic. Here's an example:

```
Begin_Com Role(*EXTENDS #PRIM_WAM)
  Web_Map For(*none) Fields(#LOGGEDON) Options(*PERSIST)
  Webroutine Name(Logon)
    Web_Map For(*input) Fields(#USER #PASSWORD)
    Define_Com Class(#ADHLOGON) Name(#UserServices)
    If (#UserServices.ValidateLogOnDetails( #USER, #PASSWORD ))
      #LOGGEDON := True
      Transfer Toroutine(LogOnSuccessful)
    Else
      Transfer Toroutine(LogOnFailed)
    Endif
  Endroutine
  Webroutine Name(LogOnSuccessful)
  Endroutine
  Webroutine Name(LogOnFailed)
  Endroutine
End_Com
```

The *Logon* Webroutine accepts a user id and password from some other Webroutine that is displayed at the Presentation Layer. An instance of the Reusable Part ADHLOGON comes alive and its *ValidateLogOnDetails* method is used to validate the user id and password. If successful, the #LOGGEDON field (declared as persistent session data) is set to True and control is transferred to the *LogOnSuccessful* Webroutine. If logon is unsuccessful, control is

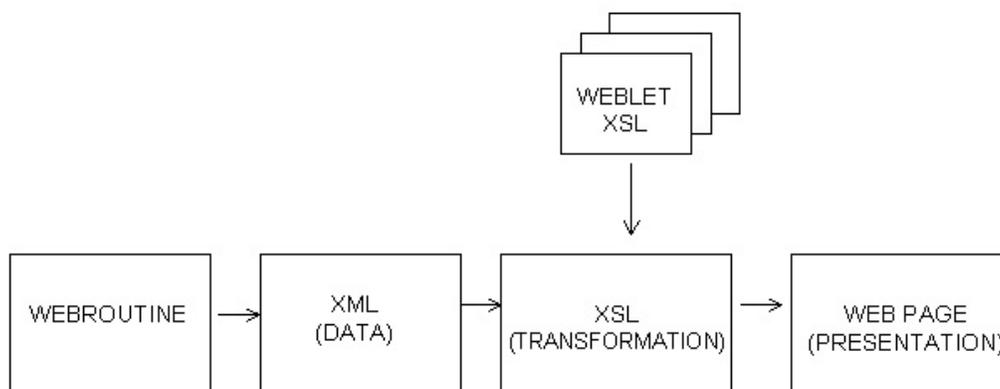
transferred to the *LogOnFailed* Webroutine.

As you can see, this is a great technique to ensure your Application Logic is encapsulated in a single place so that it can be used over and over by other WAMs.

## 1.5 The Presentation Layer

### What is the Presentation Layer?

The following diagram shows the path from a Webroutine to the "glass" of the device the user is looking at. The Presentation Layer consists of everything to the right of the Webroutine box.



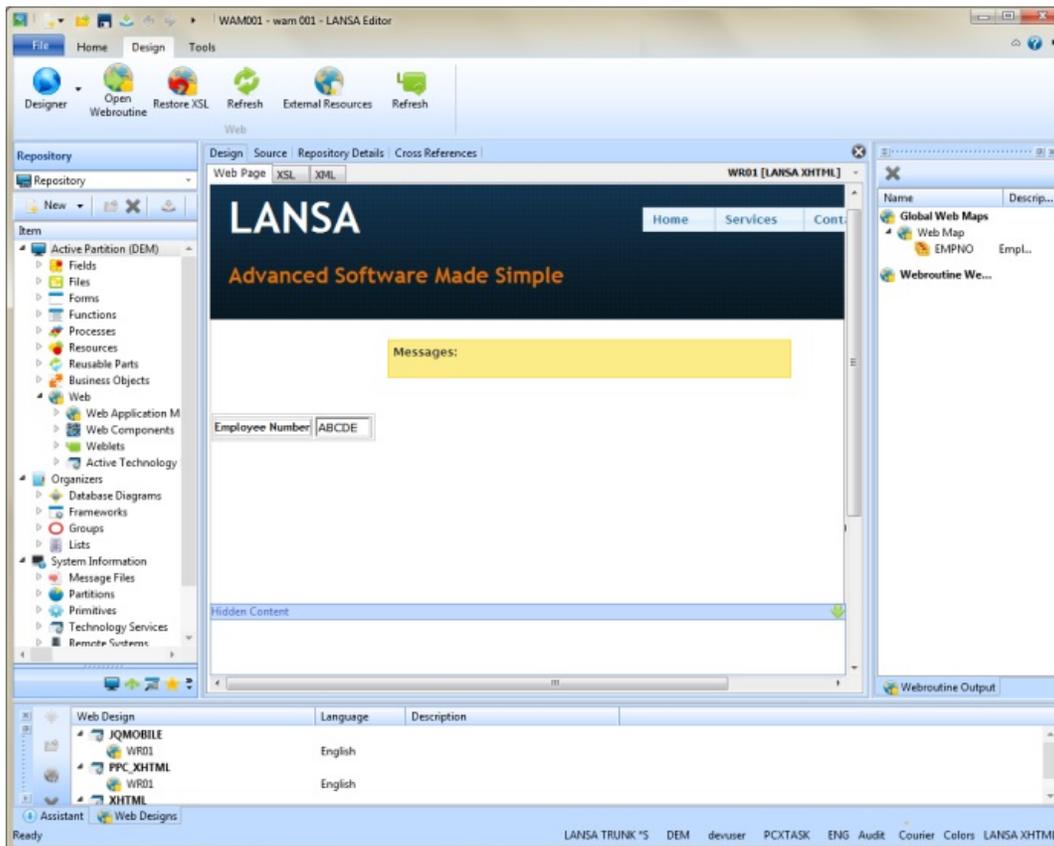
When the Webroutine ends, field and working list data (as defined on *\*output* and *\*both* Web Maps) is written out to an XML document. This is merged with the XSL generated for the User Interface (including the XSL of any Weblets being used) and then transformed into a document that is displayed to the user (in this instance, a web page in the form of XHTML).

## 1.5.1 The Editor

The area of the LANSA Editor where a Web Design is maintained is a visual editor: you use it to 'paint' your web designs. Any WAM that you have open in the Source tab of the LANSA Editor can have its Weboutines' web pages designed in the Design tab.

To open a Web Design for a WAM's Webroutine, select the green arrow, also known as the Webroutine Design Glyph, immediately to the right of the Webroutine command. If the Webroutine does not have a Web Design for the active Technology Service Provider, one will be automatically generated. Otherwise the Design tab will load with the selected Webroutine's Web Design for the current Technology Service Provider.

The following composite graphic shows what it might look like:



The Design tab's *Web Page* is the visual display of the XSL source that is used to generate the web page that the user sees. You can look at the actual, underlying XSL, by selecting the *XSL* tab. Similarly, the *XML* can be viewed by clicking that tab.

A few additional tabs that act as aids and detailers in the page-painting process are also available. These tabs can be arranged to the left, right or bottom of the Editor's window, or they may be free floating. For details about the LANSA Editor's main features, refer to [Setting up Your Workspace](#) in the *User Guide*.

In the main Editor's pane, as shown above are:

- The *Outline* tab shows fields and lists, as well as other elements contained in the Web Page currently being edited. Clicking on an entry in this list selects the corresponding control in the Web Page tab. If a drag and drop operation is hovering over an HTML or XSL element in the design, that element will be highlighted in the *Outline* tab to give a visual feedback and assist with drag and drop operations.
- The *Web Design* tab showing all Web Designs for all languages and all Technology Service Providers for the current Webroutine. This tab is useful for deleting Web Designs, rolling back changes, creating language copies and so on.

In the left hand pane in the graphic, you will see:

- The *Repository* tab that lists all the available objects that are stored in the Repository. LANSA Fields can be dragged and dropped (either as a field or as a list) from the Repository to the Web Page. You can also drag and drop from the *Repository* into the *Webroutine Output* tab, and from *Webroutine Output* tab to the Web Page.
- The *Details* tab shows a Properties for the control that is currently selected in the Web Page tab.
- The *Webroutine Output* tab shows fields and lists that are in the Web Maps available to the current Webroutine. You can drag and drop directly from this list to the Web Page, or from the *Repository* to this tab, if required.

You would use the *WebRoutine Output* tab when you want to:

change the order of fields in a list before dropping the list on the design. It cannot be done elsewhere.

move fields and lists between web maps, moving them from local web maps to global web maps and vice versa.

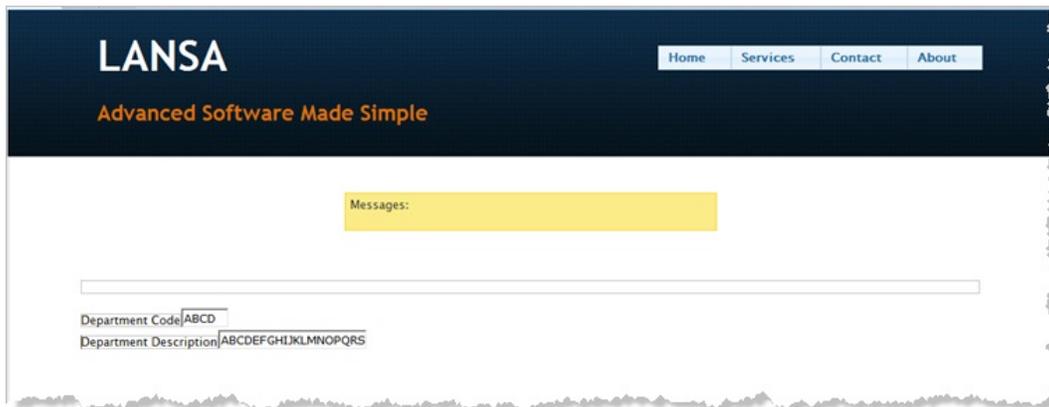
To move objects between the Repository and the *WebRoutine Output* tab, you would organize your screen layout to show both tabs at the same time.

## 1.5.2 What is Generated for the Presentation Layer by Default?

When a WAM is compiled, some default User Interface pieces are built, based on the Web Maps specified. These are different depending on whether fields or working lists are being mapped. For example, this Webroutine maps two fields:

```
Webroutine Name(MyWebroutine)
  Web_Map For(*both) Fields((#DEPARTMENT *output) #DEPTDESC)
  * some code to fetch the department description from a file here.
Endroutine
```

and will generate a User Interface in the form of a web page that looks something like this:



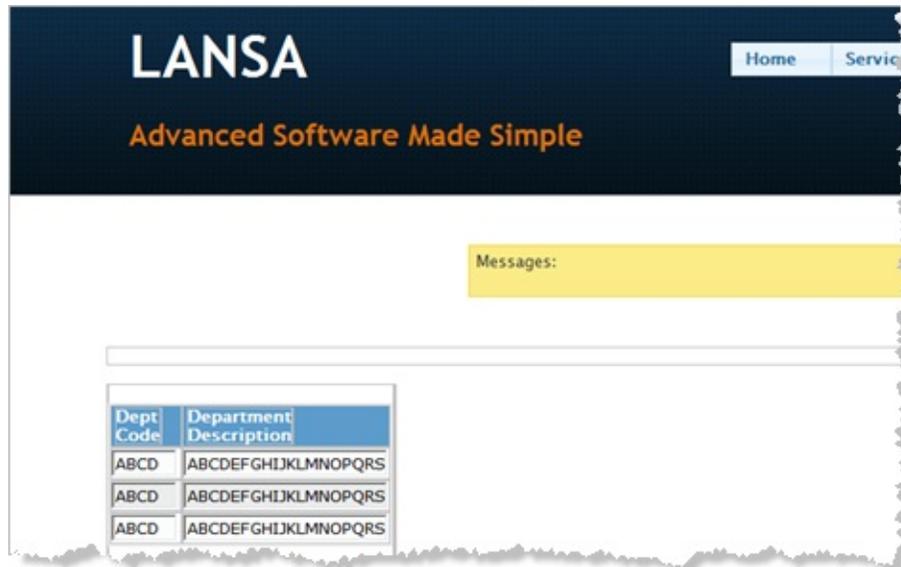
As you can see, a table has been generated with two columns and a row for each field. The *\*output* attribute on DEPARTMENT means that it cannot be modified by the user, whilst the implied attribute of *\*input* for DEPTDESC means that it is input-capable.

For this Webroutine, that specifies a working list:

```
Def_List Name(#WLDEPTS) Fields(#DEPARTMENT #DEPTDESC) Type(*Working)

Webroutine Name(MyWebroutine)
  Web_Map For(*both) Fields(#WLDEPTS)
  * some code to fetch the department description from a file here.
Endroutine
```

a web page will be generated that looks something like this:



A table has been generated with two columns, each representing a field in the list, and with each row of the table representing an entry in the list. By default, three rows of the table are shown for design purposes.

These default pieces of User Interface are all well and good, but what if you want to do something a bit more 'webby', such as visualize fields as clickable images or hyperlinks? This is where [1.5.3 Weblets](#) come in.

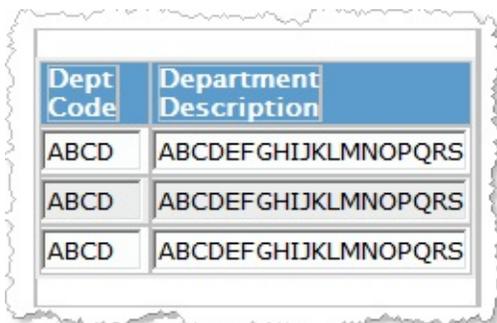
### 1.5.3 Weblets

Weblets are pieces of XSL that can take information about Fields and Working Lists and present them in different ways.

At this introductory level, think of Weblets as being visual building blocks with which the user interacts on the web page. Weblets provide hyperlinks, push buttons, clickable images and so on. Often, they can be used to visualize fields and working lists.

For example, you might want the user to be able to click on a Department Code or description of their choosing to perform some action. By dragging and dropping the Anchor Weblet onto the columns in the LANSA Editor, the Dept Code and Department Description are displayed as hyperlinks, causing:

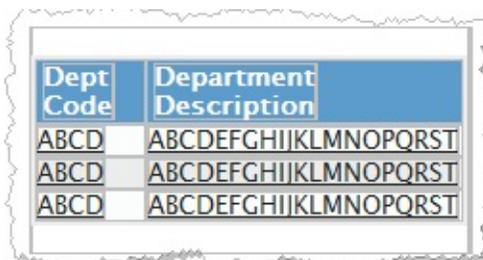
this:



A table with two columns: 'Dept Code' and 'Department Description'. The header row has a blue background. Below the header are three rows of data, each with 'ABCD' in the first column and 'ABCDEFGHIJKLMNOPS' in the second column.

Dept Code	Department Description
ABCD	ABCDEFGHIJKLMNOPS
ABCD	ABCDEFGHIJKLMNOPS
ABCD	ABCDEFGHIJKLMNOPS

to become this:



A table with two columns: 'Dept Code' and 'Department Description'. The header row has a blue background. Below the header are three rows of data, each with 'ABCD' in the first column and 'ABCDEFGHIJKLMNOPS' in the second column. The text in the data rows is underlined, indicating it is a hyperlink.

Dept Code	Department Description
<u>ABCD</u>	<u>ABCDEFGHIJKLMNOPS</u>
<u>ABCD</u>	<u>ABCDEFGHIJKLMNOPS</u>
<u>ABCD</u>	<u>ABCDEFGHIJKLMNOPS</u>

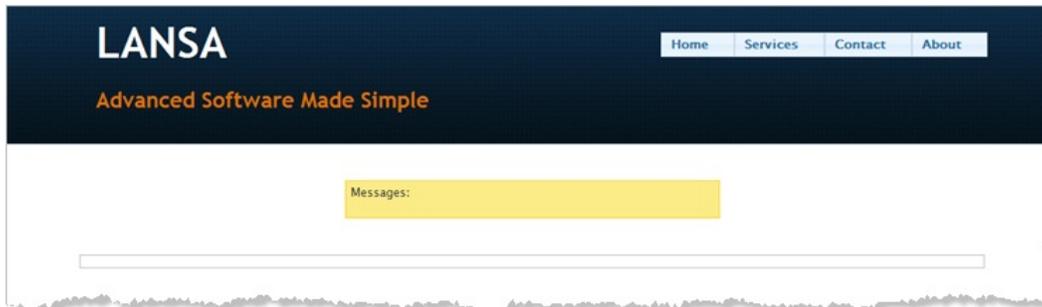
Following is a list of some of the Weblets that are shipped as standard with Visual LANSA:

Item	Description
Anchor	Standard Hyperlink
Attachment Panel	Panel with attachment layout manager
Banner	Standard Banner
Dynamic HTML menu bar	Standard DHTML Menu
Grid	Standard Grid
Horizontal splitter	Standard horizontal splitter
Large List	Standard Large List
List paging buttons	Standard List Paging Buttons
List paging images	Standard List Paging Images
Listbox	Standard listbox
Memo using a field	Standard Textarea
Memo using a list	Textarea from a list
Menu item	Standard Menu Item
Messages	Standard Messages
Navigation panel	Navigable panel
Panel	Standard Panel
Prompter	Standard Prompter
Push button	Standard Button
Push button with images	Standard Image Button
Radio button	Standard Radio Button
Radio group	Static radio buttons
Standard basic layout	Standard Layout
Standard hidden fields	Standard Hidden Fields
Standard layout schema #1	Standard Layout Schema #1
Standard layout schema #2	Standard Layout Schema #2
Standard layout schema #3	Standard Layout Schema #3
Standard layout schema #4	Standard Layout Schema #4
Standard layout schema #5	Standard Layout Schema #5
Tree view	Standard treeview control IE only
Tree view target	Standard treeview control IE only
Vertical splitter	Standard vertical splitter

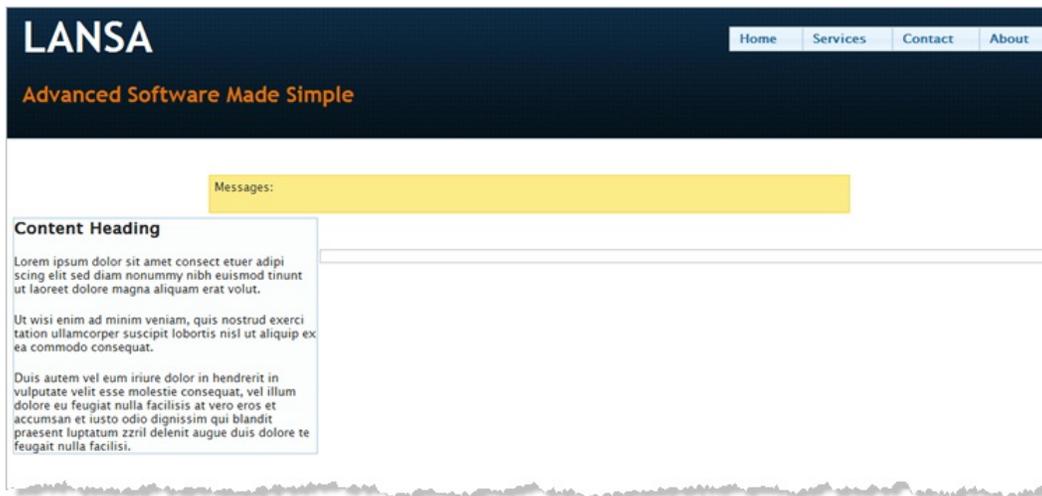
Along with the more recognizable types of User Interface widget, such as list boxes and push buttons, you will see some 'standard layout schemas'. These Weblets give your web pages a basic look and feel and provide:

- a place to display a company logo
- a framework for various menus
- a place where application messages get sent.

Standard layout schema 1, for example, is the one you've seen many times elsewhere in this document. It looks like this:



Standard layout schema 2 looks a bit different - it has a left aligned column, for instance:



Weblets, just like standard Visual LANSA User Interface controls, have properties to control their look and behavior. These are shown on the Property tab, which you access by clicking on the *Details* tab when the Weblet is selected. Following is the Property Sheet for the Anchor Weblet:

Properties	
<xsl:call-template>	
Name	Value
name	std_anchor
With Parameters	
name	\$DEPARTMENT/@id
value	\$DEPARTMENT
currentrowhfield	STDROWNUM
currentrownumval	position()
reentryfield	STDREENTRY
reentryvalue	D
hide_if	False
formname	LANSA
url	javascript:void();
on_click_wamname	\$!web_WAMName
on_click_wrname	
protocol	
show_in_new_window	False
target_window_name	<xsl:if xmlns:xsl="http://www.w3.org/199
pos_absolute_design	
width_design	
relative-image-path	"
absolute-image-path	<xsl:if xmlns:xsl="http://www.w3.org/199
class	std_anchor
mouseover_class	
text_class	std_anchor_text
presubmit_js	
tab_index	

Note the name and value parameters/properties near the top of the sheet. The name of the Weblet instance is the same name as the LANSA field it's visualizing (DEPARTMENT) and its value is set to the field's value.

Look down the sheet at some of the other properties. The on\_click\_wamname and on\_click\_wrname properties, for instance, specify the name of the WAM and Webroutine to be invoked when the user clicks on the Anchor. Back in MYWAM01, the *DepartmentSelected* Webroutine might look something like this:

```

Webroutine Name(DepartmentSelected)
  Web_Map For(*input) Fields(#DEPARTMENT)
  * do something with the selected department.
  Transfer To routine(ShowPage)
Endroutine

```

Weblets can play a non-visual role, such as supplying underlying Javascript

code or other, non-displayable components of the web page.

## 1.6 A WAM Example - Beginning to End

Let's walk through a simple two-page web example in order to reiterate what happens at key points in the execution of a WAM.

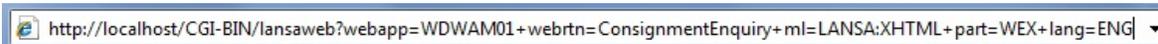
The initial page, when displayed in a browser, looks like this:



As you can see, it's a Consignment Status Enquiry. It provides for a Consignment Note Number to be entered. The Check Consignment button can then be clicked to retrieve the information from the database and display the status of the Consignment. So, we have three steps. To keep it simple, we'll have a Webroutine for each step:

- one to show the initial page, as above (ConsignmentEnquiry),
- one to receive and validate the Consignment Note Number (CheckConsignment),
- and one to fetch and show the data from the database (ShowConsignment).

To begin with, how do you display the initial page? You may enter a URL:

The image shows a screenshot of a browser address bar. The URL is `http://localhost/CGI-BIN/lansaweb?webapp=WDWAM01+web rtn=ConsignmentEnquiry+ml=LANSA:XHTML+part=WEX+lang=ENG`. The address bar has a small icon on the left and a dropdown arrow on the right.

Note the **webapp=WDWAM01** and **web rtn=ConsignmentEnquiry** parameters. These indicate the name of the WAM (**WDWAM01**) and Webroutine (**ConsignmentEnquiry**) to be executed.

Or you may execute this from some other part of a bigger WAM application, perhaps by clicking on an Anchor (or Hyperlink), by pushing a button or by selecting a menu item. Regardless of the method, the mechanism is the same: a Webroutine, sitting inside a WAM, is executed.

So, what does the underlying code for our initial *ConsignmentEnquiry* Webroutine look like? Here it is:

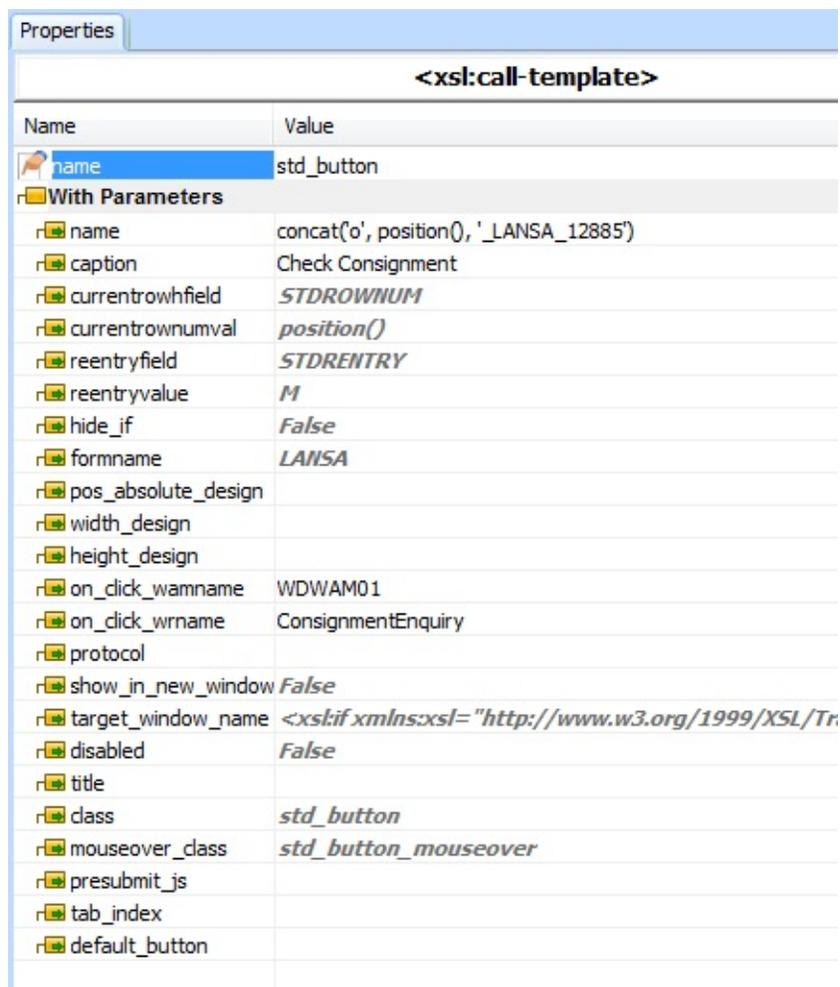
```

Webroutine Name(ConsignmentEnquiry) Desc('ACME Couriers Consignment Status Enquiry')
  Web_Map For(*both) Fields(#WDCONSIGN)
Endroutine

```

The key here is the web map for the WDCONSIGN field. It is defined as *\*both*, meaning that its value will be read in at the start of the Webroutine and written out when the Webroutine ends. Remember, when you compile the WAM, LANSA will create some XSL to display WDCONSIGN on the web page.

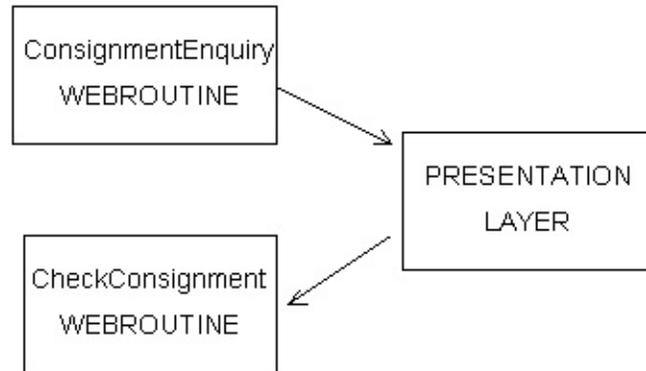
It won't, however, give you a push button. So, you right-click on the Webroutine and open its page in the LANSA Editor. Drag and drop a push button Weblet onto the design and set its properties:



<xsl:call-template>	
Name	Value
name	std_button
With Parameters	
name	concat('o', position(), '_LANSA_12885')
caption	Check Consignment
currentrowhfield	STDROWNUM
currentrownumval	position()
reentryfield	STDREENTRY
reentryvalue	M
hide_if	False
formname	LANSA
pos_absolute_design	
width_design	
height_design	
on_click_wamname	WDWAM01
on_click_wrname	ConsignmentEnquiry
protocol	
show_in_new_window	False
target_window_name	<xsl:if xmlns:xsl="http://www.w3.org/1999/XSL/Tr
disabled	False
title	
class	std_button
mouseover_class	std_button_mouseover
presubmit_js	
tab_index	
default_button	

Note the **caption**, **on\_click\_wamname** and **on\_click\_wrname** property settings. The on\_click properties executes the *CheckConsignment* Webroutine in the WDWAM01 WAM when the button is clicked.

So far, then, the flow of this little application looks like this:



Now, back in the LANSa Editor, enter the following RDMLX code for the *CheckConsignment* Webroutine:

```

Webroutine Name(CheckConsignment)
  Web_Map For(*input) Fields(#WDCONSIGN)

  * see if this consignment exists in the consignment status file.
  Check_For In_File(WDCONST) With_Key(#WDCONSIGN)

  * it exists ...
  If_Status Is(*EQUALKEY)
    * show the consignment status.
    Transfer Toroutine(ShowConsignment)
  Else
    * it doesn't exist: show an error message ...
    Message Msgtxt('Invalid Consignment Note number. Please try again.')

    * and re-display the main page.
    Transfer Toroutine(ConsignmentEnquiry)
  Endif
Endroutine
  
```

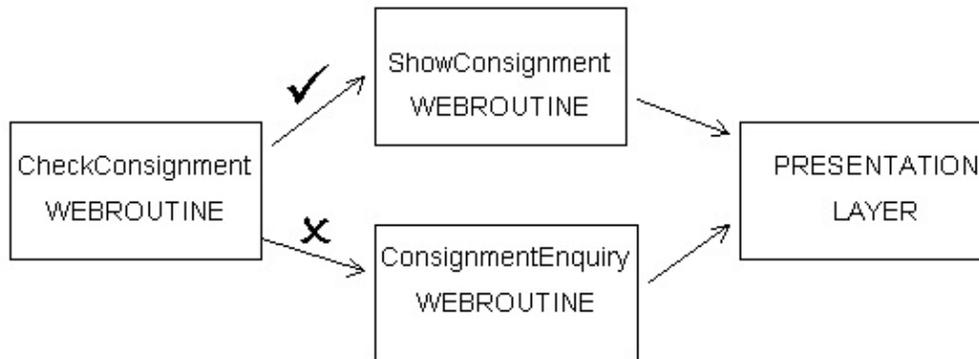
Again, note the web map. Its For(\*input) setting means that WDCONSIGN is mapped into the Webroutine when it's invoked – by the user clicking the push button, in this instance.

The Consignment status file is checked to see if the entered Consignment Note Number exists. If it does exist, control is transferred to the *ShowConsignment* Webroutine. If it doesn't exist, an appropriate error message is issued and control is transferred to the original *ConsignmentEnquiry* Webroutine.

Note that the *CheckConsignment* Webroutine never ends, and so never displays a web page. It always transfers control to either the *ShowConsignment* or

### ConsignmentEnquiry Webroutines.

The flow for this part of the WAM, then, looks like this:



Now have a look at the code for the *ShowConsignment* Webroutine:

```
Webroutine Name>ShowConsignment Desc('ACME Couriers Consignment Status Enquiry')
  Web_Map For(*both) Fields((#WDCONSIGN *output))
  Web_Map For(*output) Fields((#WDCONSTS *output))

  * fetch the consignment status from the file.
  Fetch Fields(#WDCONSTS) From_File(WDCONST) With_Key(#WDCONSIGN)
Endroutine
```

Again, note the Web Maps. The Consignment Note Number (WDCONSIGN) comes in courtesy of the *\*both* web map.

The status field, WDCONSTS, is fetched from the database using the Consignment Note Number as a key.

The Webroutine ends. The **Consignment Note Number** and **Status** fields are mapped out of the Webroutine.

Control is transferred to the Presentation Layer, which displays the Consignment Note Number and Status. Note too the *\*output* attributes on the fields, which means they will be displayed as output-only.

Let's run the WAM from the beginning, assuming a valid Consignment Note Number has been entered:

http://localhost/CGI-BIN/lansaweb?webapp=WDWAM01+webtrn=ConsignmentEnquiry+ml=LANSA:XHTML+part=WEX+lang=ENG

```

Webroutine Name(ConsignmentEnquiry) Desc('ACME Consignment Status Enquiry')
Web_Map For(*BOTH) Fields(#WDCONSIGN)
Endroutine

```

LANSA  
Advanced Software Made Simple

Home Services Contact About

ACME Consignment Status Enquiry

Consignment Note Number A1234567890

```

Webroutine Name(CheckConsignment)
Web_Map For(*input) Fields(#WDCONSIGN)
* validation code here
* show the consignment status
Transfer Toroutine(ShowConsignment)
Endroutine

```

```

Webroutine Name(ShowConsignment) Desc('ACME Consignment Status Enquiry')
Web_Map For(*BOTH) Fields((#WDCONSIGN *output))
Web_Map For(*OUTPUT) Fields((#WDCONSigs *output))
* fetch the consignment status from the file
Endroutine

```

LANSA  
Advanced Software Made Simple

Home Services Contact About

ACME Consignment Status Enquiry

Consignment Note Number A1234567890  
Consignment Status Shipped on 12/24/2010

```

Webroutine Name(ConsignmentEnquiry) Desc('ACME Consignment Status Enquiry')
Web_Map For(*BOTH) Fields(#WDCONSIGN)
Endroutine

```

Note the *Check Another* push button. It simply takes us back to where we started by re-executing the *ComponentEnquiry* Webroutine.

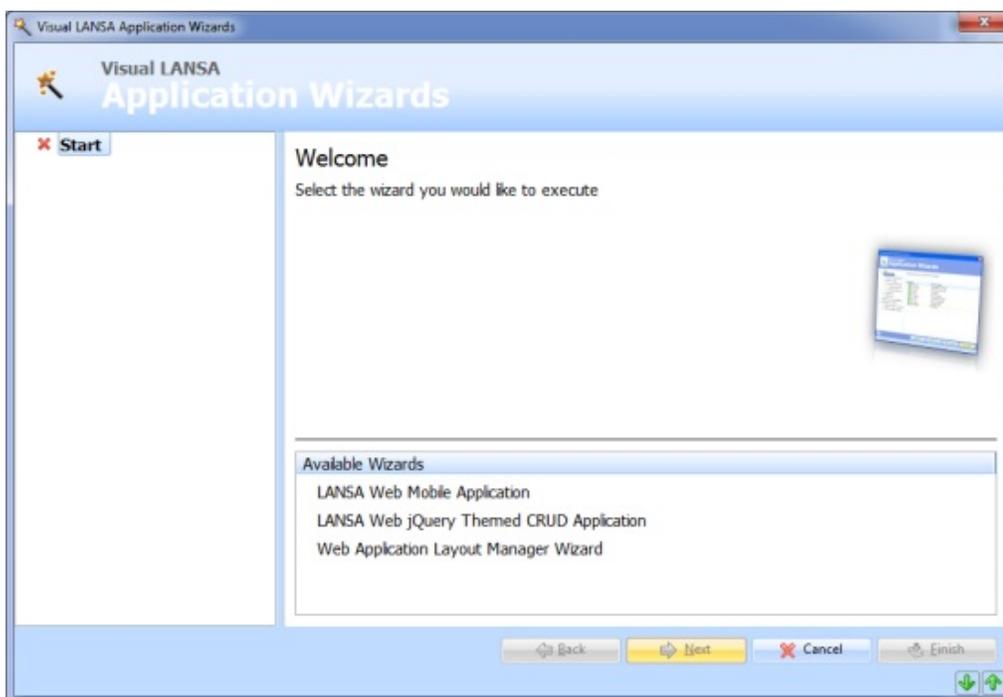
## 1.7 WAM Wizards

The *Visual LANSa Application Wizards* guide you in the creation of complex Visual LANSa applications using a series of predefined steps.

Before you can run a Visual LANSa Wizard:

- Visual LANSa needs to be installed and correctly configured
- The partition needs to be Web enabled
- The system needs to have a web server installed and configured to run WAMs if the execute option is selected.

To access the wizards, in the LANSa editor, select the Tools tab, and in the Utilities group, click on Wizards.



To start, click on any of the wizards in the *Available Wizards* list.

Each Wizard contains a list of questions which are listed with a ✘, once you answer each question, the ✘ becomes a ✔. You need to answer all the questions before you can the *Finish* the wizard.

You can navigate between the different questions by using the back  and next  buttons.

The up  and down  arrows at the bottom right corner of the wizard dialog are used to scroll through the messages, if any messages are displayed in the message bar.

The Wizards available are:

- [1.7.1 LANSa Web Mobile Application](#) This wizard generates LANSa Web Mobile Browser applications with the ability to filter, view lists and show Header/Detailer forms including a Sampler to showcase the types of controls that can be used within your own LANSa Web Mobile applications.
- [1.7.2 LANSa Web jQuery Themed CRUD Application](#)  
This wizard generates a complete LANSa for the Web CRUD (Create, Read, Update, Delete) application over any LANSa or keyed files.  
Features include:

jQuery themed using your selected site layout or the default

Drilldown to related files

Searches over application data.

- [1.7.3 Web Application Layout Manager Wizard](#)  
This wizard provides an easy way to customise and generate the site layout for your web application. Features include:

Colour scheme and look and feel

Multiple content areas

Ability to use both LANSa weblets and jQuery weblets

Enable Web 2.0 sites using AJAX.

## 1.7.1 LANSA Web Mobile Application

This wizard generates a totally customisable jQuery Mobile WAM. Depending on your answers, the generated WAM will include one or more of the following:

- A **Sampler** webpage containing all the controls available to users. This page is for the whole application.
- A user defined **Webpage** using one or more of the following:

Heading

Text Block

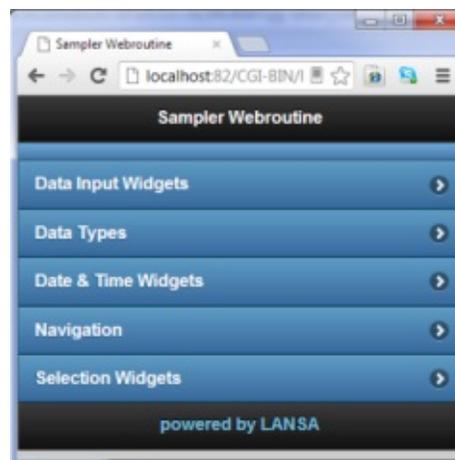
Image

Form elements. This element contains a collection of popular form controls.

Link

List view. This control can contain one or more of the following:

- List Data, which is data dynamically loaded from RDML. There is only one list data per list. If you add another list control then you can specify another List Data.
- Divider
- Static Menu Item, which will link to one of the previously created webpages.



[↑ 1.7 WAM Wizards](#)

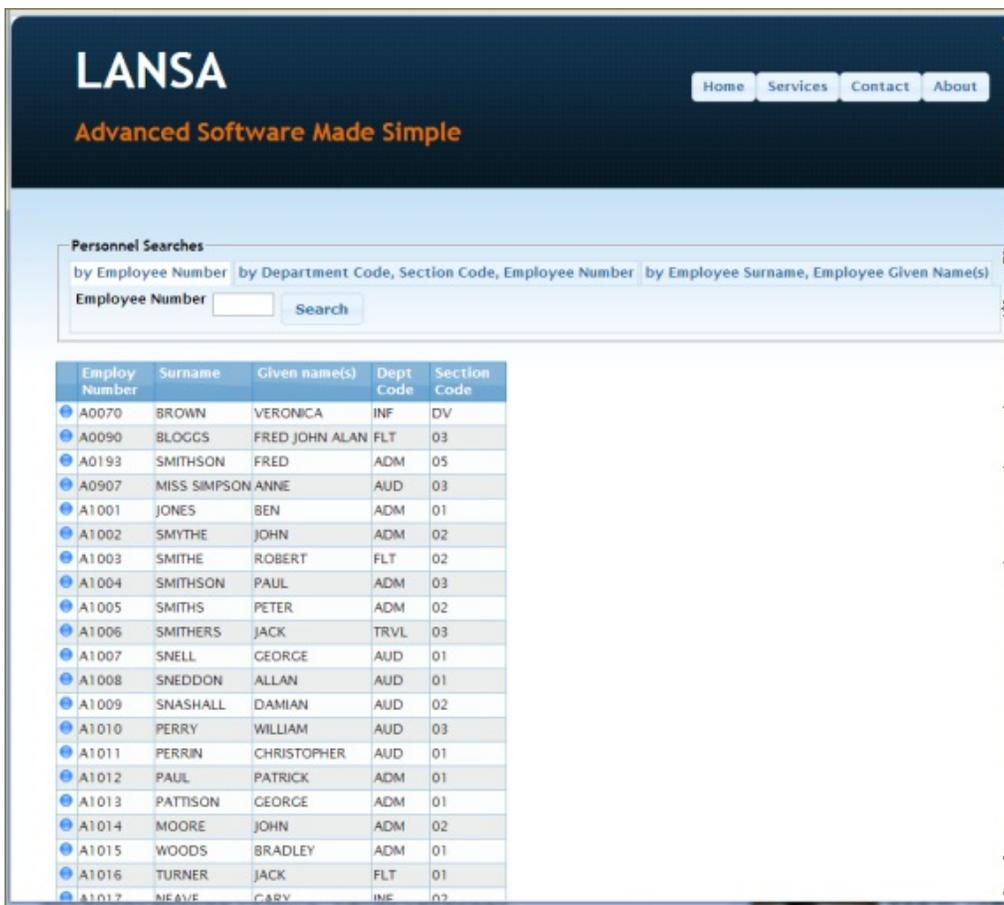
## 1.7.2 LANSA Web jQuery Themed CRUD Application

This wizard generates a WAM and a WAM layout to perform file maintenance operations.

You can choose to restrict the generated WAM to *Read only* operations or you can authorise *Create*, *Update* or *Delete* operations.

The fields displayed on the *Add* and *ShowRecord* pages as well as the columns on the *SearchList* page are customizable. You can drill down to another WAM provided it matches the *AccessRoute* details.

Following is an example of the wizard with file PSLMST:



The screenshot displays the LANSA web application interface. At the top, the LANSA logo is visible on the left, and navigation buttons for Home, Services, Contact, and About are on the right. Below the logo, the tagline "Advanced Software Made Simple" is shown. The main content area is titled "Personnel Searches" and features a search form with three radio button options: "by Employee Number", "by Department Code, Section Code, Employee Number", and "by Employee Surname, Employee Given Name(s)". The "by Employee Number" option is selected. Below the radio buttons is an input field for "Employee Number" and a "Search" button. Below the search form is a table listing employee records.

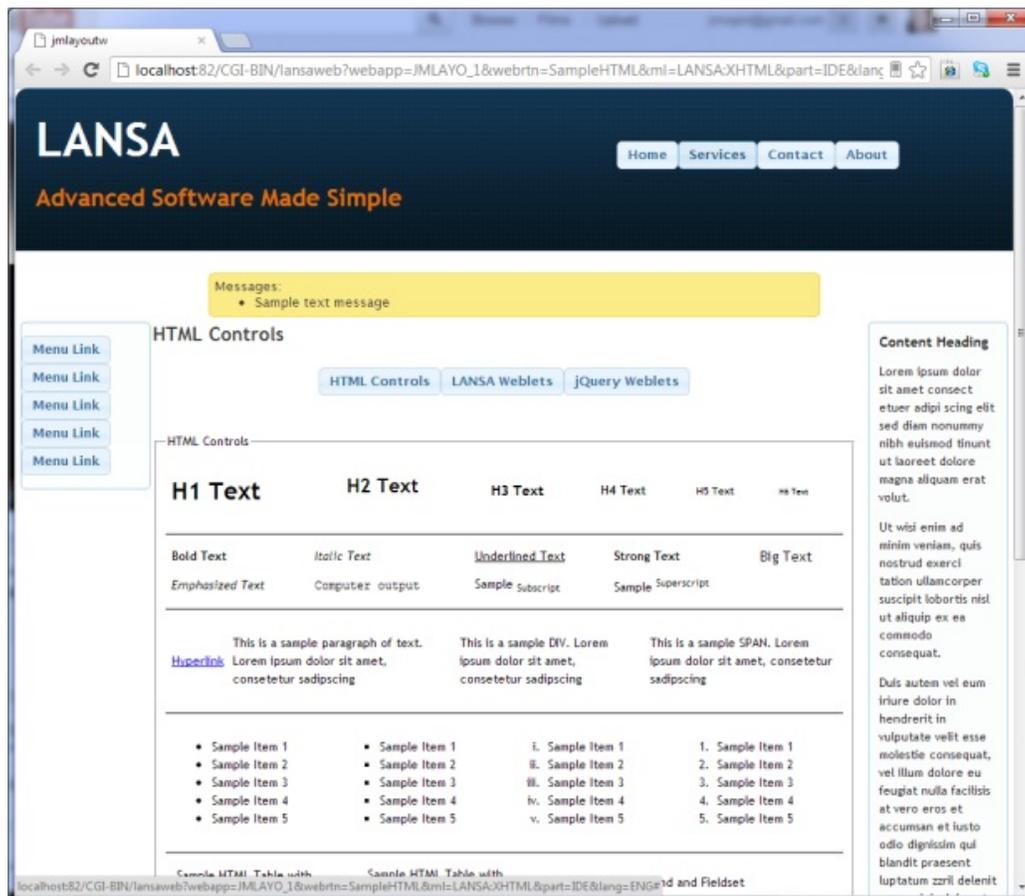
Employ Number	Surname	Given name(s)	Dept Code	Section Code
A0070	BROWN	VERONICA	INF	DV
A0090	BLOGGS	FRED JOHN ALAN	FLT	03
A0193	SMITHSON	FRED	ADM	05
A0907	MISS SIMPSON ANNE		AUD	03
A1001	JONES	BEN	ADM	01
A1002	SMYTHE	JOHN	ADM	02
A1003	SMITHE	ROBERT	FLT	02
A1004	SMITHSON	PAUL	ADM	03
A1005	SMITHS	PETER	ADM	02
A1006	SMITHERS	JACK	TRVL	03
A1007	SNELL	GEORGE	AUD	01
A1008	SNEDDON	ALLAN	AUD	01
A1009	SNASHALL	DAMIAN	AUD	02
A1010	PERRY	WILLIAM	AUD	03
A1011	PERRIN	CHRISTOPHER	AUD	01
A1012	PAUL	PATRICK	ADM	01
A1013	PATTISON	GEORGE	ADM	01
A1014	MOORE	JOHN	ADM	02
A1015	WOODS	BRADLEY	ADM	01
A1016	TURNER	JACK	FLT	01
A1017	NEAVE	CARY	INF	03

↑ 1.7 WAM Wizards

## 1.7.3 Web Application Layout Manager Wizard

Depending on the answers the user provides, the wizard will generate a layout weblet to the requirements you have specified, and a subfolder in the images folder of the webserver. The following might be included depending on the answers provided:

- A CSS file containing the styles in the webserver images subfolder with its matching external resource object.
- A JavaScript file containing additional JavaScript code in the webserver images subfolder and its matching external resource object.
- A sample WAM using the generated layout. In this case a WAM layout is also created to reference the generated layout weblet.



Refer to WAM025 - Using the Layout Wizard to step through the Wizard.

↑ 1.7 WAM Wizards



## **2. WAMs Deconstructed**

### **Prerequisite Reading**

Before reading this chapter, we recommend that you read [An Introduction to WAMs](#) for an overview of LANSAs WAM Architecture.

## 2.1 The Relationship Between WAMs, Webroutines, Weblet and Weblet Templates

After reading [An Introduction to WAMs](#) you should have a clear picture of what WAMs and webroutines are, so we'll now shift the focus and explain, in more depth, weblets and weblet templates, and how they interact with WAMs and webroutines to build up and customize the presentation layer.

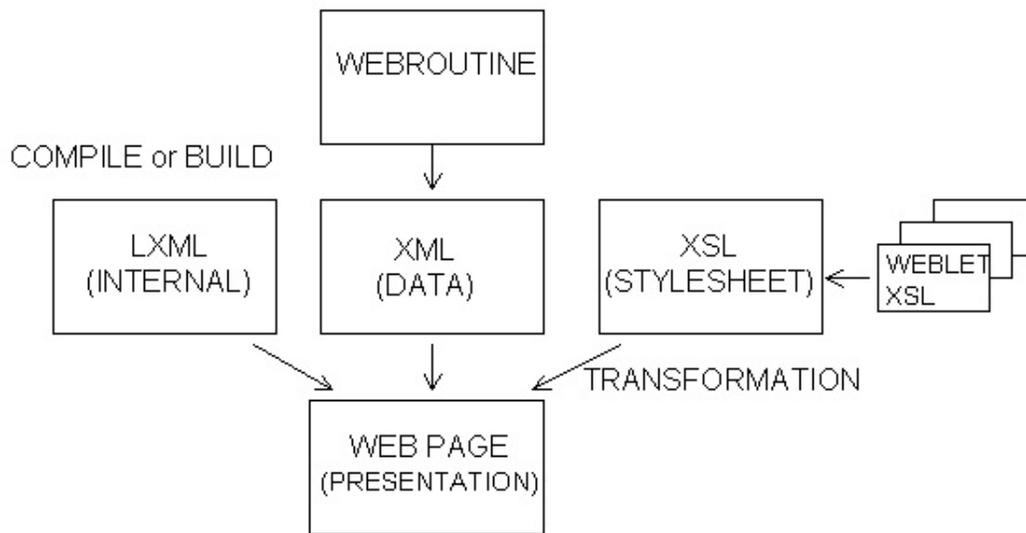
Let's review some WAM concepts. You should already be familiar with most of these concepts:

- Simplistically, each webroutine is comprised of two parts: the RDMLX portion encapsulating the application logic (in the Application Logic Layer), and the XSL/XML portion defining the presentation interface (in the Presentation Layer, also known as the Web Design).
- A WAM includes one or more webroutines.
- All webroutines are defined in the RDMLX code of a WAM using a simple WEBROUTINE / ENDROUTINE construct. The webroutine definition may include WEB\_MAP commands to pass field and working list information to and from the presentation layer.
- The RDMLX code of a WAM can also include method routines (MTHROUTINE) and subroutines (SUBROUTINE). If you need a refresher on these concepts refer to the LANSAs Technical Reference Guide.
- Webroutines are generally presented as a user interface (for example, HTML for a web browser), but can be generated as a non-visual presentation if the appropriate Technology Services are defined.
- Typically, opening a webroutine in the LANSAs Editor opens the Extensible Stylesheet (XSL) object generated for the webroutine so it can be viewed and modified as required. The initial presentation in the LANSAs Editor is based on the internally defined LXML (a list representation of XML tags) and the XSL/XML objects, which can be generated during a build. We'll review these objects again later in this document.
- A weblet includes one or more weblet templates. Weblet Templates determine how a weblet is applied for a technology service or another variant which requires different XSL, like inline lists.
- The presentation of a webroutine can be modified by dragging and

dropping weblet templates onto the webroutine's Web Page tab in the LANSAs Editor.

- Weblet Templates are Technology Service specific. This means that while a consistent set of weblets is shown in your LANSAs repository, when you change your current Technology Service the list of available Weblet Template will be modified.

The following diagram shows the relationship between a webroutine (the application logic) and the generated XSL, XML and various weblets (the presentation layer):



## 2.1.1 What happens when I build or compile a WAM?

Let's review the process of building or compiling a WAM to see how this impacts the development process.

The easiest way to see what happens when a WAM is built or compiled is to create a simple WAM and review the resulting objects.

Let's use a very simple WAM (named KWAM10) with a single webroutine (KWAM1001):

```
FUNCTION OPTIONS(*DIRECT)
  BEGIN_COM ROLE(*EXTENDS #PRIM_WAM)
    WEBROUTINE NAME(kwam1001)
      WEB_MAP FOR(*both) FIELDS(#deptment)
    ENDROUTINE
  END_COM
```

The first time a Web Design is generated, the appropriate layout objects will be generated too. Any subsequent generation will NOT regenerate the layout objects. All changes to the generated layout are performed using the LANSA Editor.

Don't dwell on this too long as layouts will be explained in great detail later in this document.

The objects associated with the WAM layout are:

- A single layout variables object is created for the WAM – kwam10\_layout.variables.xml
- A single layout XSL stylesheet is created for the WAM – kwam10\_layout.xsl

Additional XML and XSL objects are generated or regenerated for each webroutine during the build or compile phase if you select the appropriate Generate XSL options in conjunction with one or more Technology Services. These XML and XSL objects can also be generated or regenerated for a specific webroutine when you explicitly ask them to be using the small green arrow immediately to the right of the RDMLX WEBROUTINE command. This set of objects will be created for each Partition Language and each selected Technology Service combination using the directory structure ...\\X\_WIN95\\X\_LANSA\\X\_<Partition>\\web\\<Provider>\\<Technology Service>\\<language>.

The objects associated with the Generate XSL settings are:

- An XSL stylesheet is created for each webroutine – kwam10.kwam1001.xsl
- A variables document is created for each webroutine. (This is related to the XSL object and is independent of the LXML information stored internally.) – kwam10.kwam1001.variables.xml

If you perform a build, the default system settings, to Generate XSL for all New Webroutines (that is, webroutines that have not previously had XSL/XML objects generated), will be applied. The compile options for a WAM allow more control of this process.

When compiling a WAM, the Generate XSL options allow for the generation of XSL to be bypassed, or XSL can be generated for All Webroutines or only for New Webroutines. You can also generate XSL for a single webroutine on demand using the small green arrow immediately to the right of the RDMLX WEBROUTINE command



It is important to remember that by selecting Generate XSL for All Webroutines you will regenerate the XSL and in doing so will lose any modifications applied in the LANSAs Editor. The same applies when regenerating XSL on demand.



If you are doing Multilingual Development...  
The XSL objects associated with the default partition language are published in the LANSAs repository and replicated for other languages. This process allows you to effectively have a single set of XSL information for each Technology Service for all languages. While it is possible to have a different set of XSL published for each language, this approach is generally not recommended unless you require very distinct interfaces for each language. It is a better approach to use Multilingual Variables as this requires only one Web Design and hence is easier to maintain.

In addition to the XML and XSL objects generated, whenever a WAM is compiled (this does not apply for the build option) a set of RDMLX objects associated with the WAM will always be created or recreated. The same set of objects is created for a WAM as for any other RDMLX compilable object, for

example:

A dynamic link library object kwam10.dll is created in the partition execute directory

...\X\_WIN95\X\_LANSA\X\_<Partition>\execute

A program file, kwam10.pgm is created in the partition source directory ... \X\_WIN95\X\_LANSA\X\_<Partition>\source

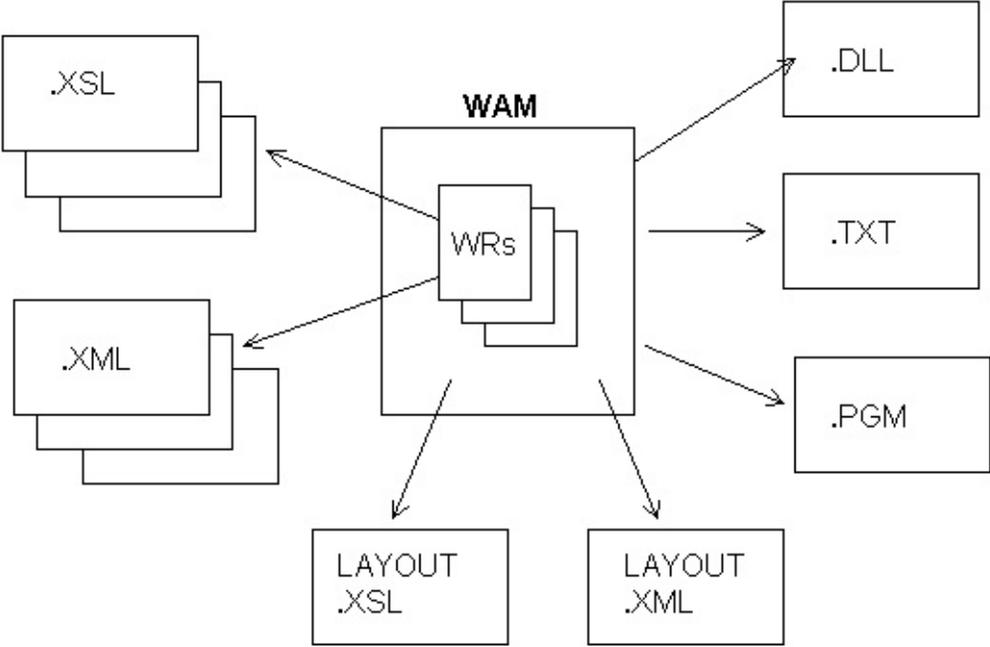
A text file, kwam10.txt is created in the partition source directory ... \X\_WIN95\X\_LANSA\X\_<Partition>\source

That covers the files generated to support a WAM, but there is one final piece required to complete the picture. When the WAM is built or compiled, LXML (a list representation of XML tags) information is ALWAYS generated, or regenerated, for each webroutine. This LXML information is stored internally in the LANSA database and is independent of Technology Service and Language.

Automatic regeneration of the LXML information is important as it ensures that any modifications to WEB\_MAP definitions are available in the Webroutine Output tab. The LXML can be viewed in the LANSA Editor by selecting the XML tab.

Some modifications to the generated LXML (cookies and TSML nodes added by the LANSA Editor) are retained when the LXML is regenerated.

This is a concise view of the WAM build and compile processes. As you can see, in the diagram below, some objects are generated for the WAM and apply to all the webroutines in the WAM, while other objects are generated for each webroutine (and some information is stored internally):



## 2.1.2 What are Webllets and Weblet Templates?

Webllets are snippets of XSL code designed to encapsulate common HTML functions and, in doing so, hide their complexity. Weblet Templates are an additional level of granularity within a Weblet, defining sub-sections of the weblet XSL code applicable to a specific Technology Service and other possible conditions.

All webllets are stored in the LANSAs repository and can be created, opened or modified in the LANSAs Editor. Webllets, and consequently Weblet Templates, are reusable and can be dragged and dropped onto the Web Page of any webroutine to assist in building up the desired Presentation Layer or Web Design. While you can drag and drop both Webllets and Weblet Templates to get the same result (in most cases) it is useful to get into the habit of working with Weblet Templates to ensure the selected weblet is supported with the current Technology Service. The Weblet Groupings can also make it easier to locate the appropriate weblet to be used. For example, if you are using inline list there is a Weblet Template grouping Inline Templates to identify all the webllets defined to be "inline-aware" for the current Technology Service.

LANSAs supplies a standard set of webllets. Webllets, like all XSL and XML objects in LANSAs, are Technology Service specific. Most shipped webllets are supported for both Technology Services provided by LANSAs, that is, XHTML (eXtensible Hypertext Markup Language) and JQMOBILE (jQuery Mobile).

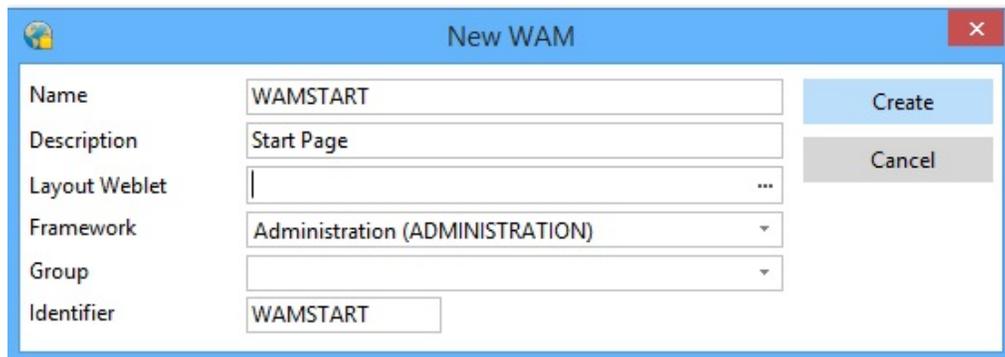
Typically webllets are used to visualize data on a web page. For example standard webllets are provided to visualize a field as a checkbox or a radio button, or a working list could be visualized as a dropdown list or a tree. There are also standard webllets provided for formatting the layout of a web page, message presentation, menus and other elements commonly included on web pages that are not specifically related to data on the interface. Additional non-visualized webllets are provided to give access to commonly required information, for example the variables and style webllets (which are described in [2.5 Variables](#) and [2.7 Cascading Style Sheets \(CSS\) and the Style Weblet](#)).

### 2.1.3 How do I use Weblets?

To get a feeling for how weblets can be used to build up the presentation layer, let's look at a simple example.

The goal in this example is to construct a simple search page to allow to a department code to be entered and then a search for department records initiated.

Create a WAM and name it WAMSTART.



Field	Value
Name	WAMSTART
Description	Start Page
Layout Weblet	...
Framework	Administration (ADMINISTRATION)
Group	
Identifier	WAMSTART

Once the WAM has been created you are presented with a dialog to create a webroutine. You can dismiss this dialog and start editing the WAM yourself, or you can use the dialog to fill in the details for your new webroutine and then go straight to the Design phase and let the RDMLX/Web Design interaction automatically build the web maps in the RDMLX source for you. In this example we will use the dialog to create a webroutine.

**New Webroutine** ✕

Enter a Webroutine name and choose Create to begin designing your Web Page

**Webroutine details**

Name

Description

Service Name

Response \*DEFAULT - return a Web Page ▾

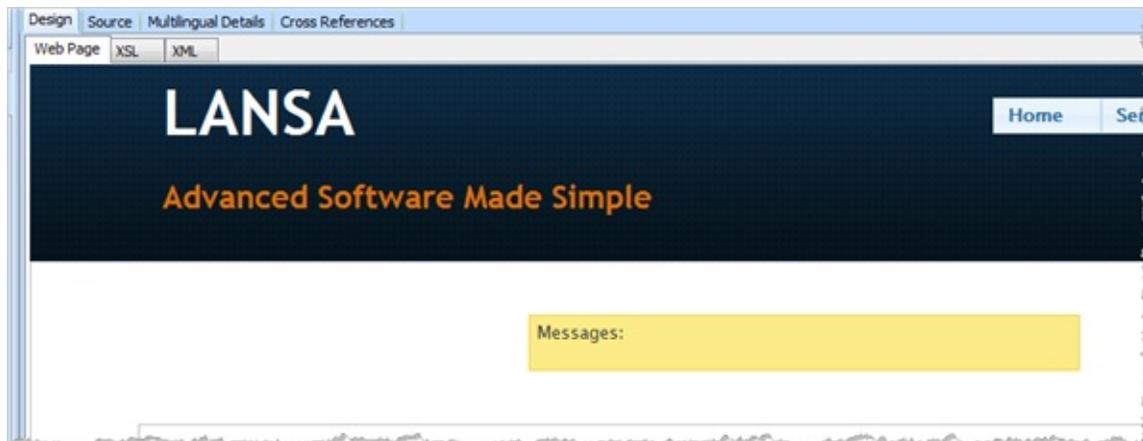
Close

**Web Page design details**

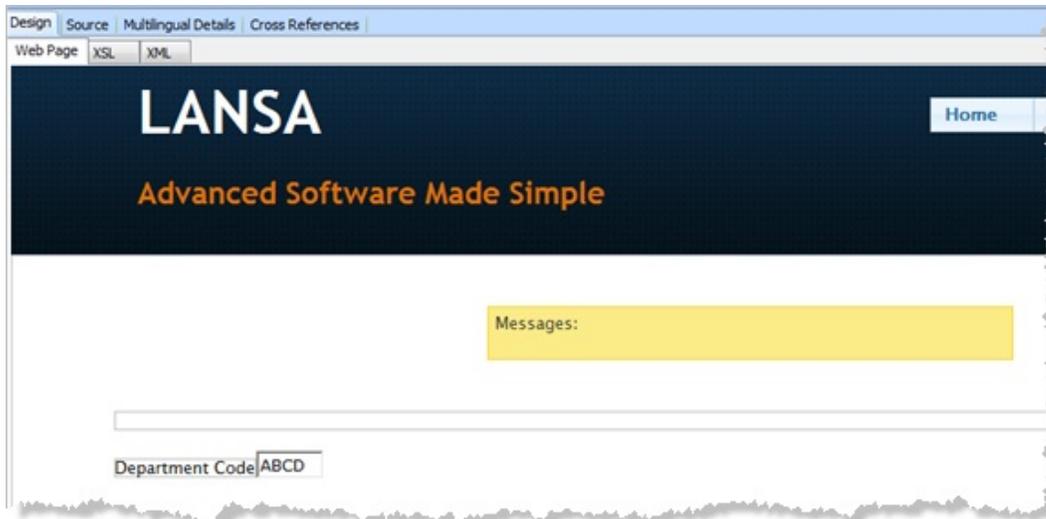
Generate XSL	Open Design	Provider	Technology Service
<input type="checkbox"/>	<input type="checkbox"/>	LANSA	JQMOBILE
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	LANSA	XHTML

Show this dialog when Web Application Module with no Webroutines is opened

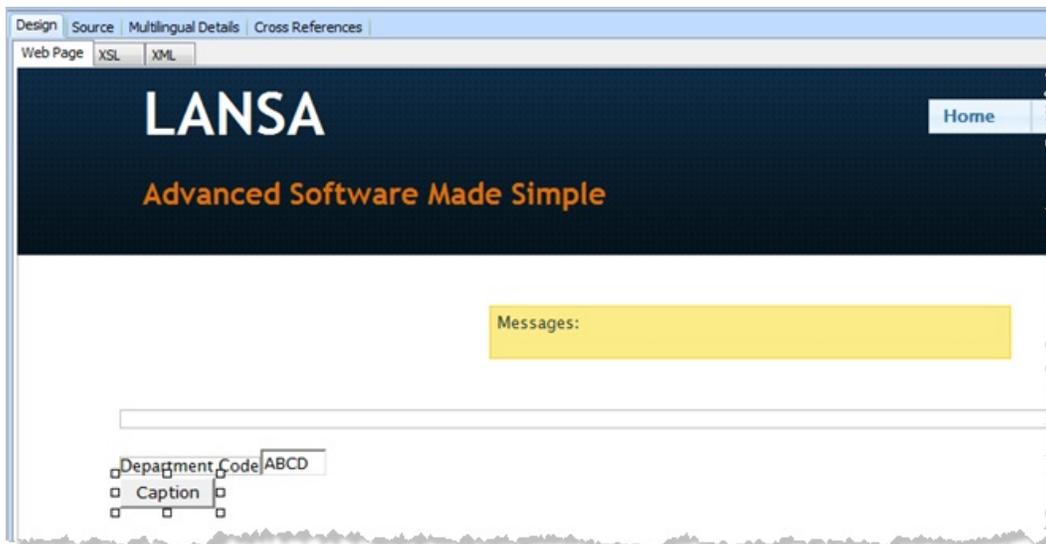
You will now automatically be taken into the Web Design phase and see the following Web Design.



Drag the DEPARTMENT field from the Repository and drop it on the Web Design. This operation will also add a web map to the RDMLX Source.



To facilitate the search you want to add a push button to the page. To do this, open the Weblet Templates in the Repository and locate the push button weblet. Simply drag the push button onto the Design view where you want it to be displayed.



Ensure the focus is on the newly added push button. Select the Details tab to review the push button's properties.

Properties	
<xsl:call-template>	
Name	Value
name	std_button
With Parameters	
name	concat('o', position(), '_LANSA_4580')
caption	Search
currentrowhfield	STDROWNUM
currentrownumval	position()
reentryfield	STDREENTRY
reentryvalue	M
hide_if	False
formname	LANSA
pos_absolute_design	
width_design	
height_design	
on_click_wamname	\$web_WAMName
on_click_wrname	deptsearch
protocol	
show_in_new_window	False
target_window_name	<xsl:if xmlns:xsl="http://www.w3.org
disabled	False
title	
class	std_button
mouseover_class	std_button_mouseover
presubmit_js	
tab_index	
default_button	

Change the caption property to Search and the on\_click\_wrname property to deptsearch. This indicates that when the push button is clicked on the resulting web page, a webroutine deptsearch (defined in the current WAM) will be executed. Note that this webroutine is currently not defined so it must be added to the WAM before the push button will execute correctly.

In this example, the appearance and functionality of the web page generated for webroutine kwam1001 is modified by dropping a field onto the design, including a reference to the push button weblet and customizing the properties as required.

Non-visualized weblets can be applied in the same way but you will need to review the XSL tab to ensure a reference to the weblet has been added as required.

Before executing the WAM it must be compiled but be careful not to regenerate the XSL for All webroutines as this will replace the manual changes to the XSL applied by dragging and dropping the weblet template.

## 2.1.4 How do I know where and when to use a Weblet?

You will be applying visualized weblets to your presentation, so the question of when and where to use a weblet is really a design consideration relating to how you want to view and modify information on the resulting web page.

For example, if you defined the following webroutine:

```
DEFINE FIELD(#yesorno) TYPE(*char) LENGTH(1) DESC('Yes or No')  
WEBROUTINE NAME(wam0190)  
  WEB_MAP FOR(*both) FIELDS(#yesorno)  
ENDROUTINE
```

then compiled the webroutine to generate a default presentation interface for XHTML, the resulting web page would look something like this:



Notice that the default representation for the mapped value is an input capable field.

By dragging the Checkbox weblet (or the Boolean field visualization weblet) onto the field value you can change the presentation of the data so it is viewed and responds as a checkbox:



 Tip – If you define the field #YESORNO as type \*BOOLEAN, the visual representation would automatically be a checkbox.

---

Refer to [Standard Field Visualizations](#) for details of the visualized weblots shipped with LANSAs, including typical usage for each weblet.

The use of non-visualized weblots requires a thorough understanding of the purpose of the respective weblet. The purpose and application of the various non-visualized weblots shipped with LANSAs is outlined later in this document.

## 2.1.5 Can I create my own Weblets?

Yes, you can create your own weblets to implement your site's standards or to encapsulate commonly used XSL code by using the New menu and choosing Weblet in the LANSAs Editor's tool bar. You can then add the appropriate XSL code to define your weblet. Typically you will not need to add XML for the weblet. The XML tab is by default not visible in the LANSAs Editor.

We recommend that you use a naming prefix other than std\_ for any weblets you create. To simplify the management of your Weblets, create your own weblet grouping, and assign this to any weblets you create.

Remember you will need to create a version of your weblet for each Technology Service you intend to use the weblet with.



Do not modify the weblets shipped with LANSAs as these will be replaced during subsequent LANSAs software upgrades.



If you create a new weblet using a standard weblet as a template, remember to change the name of the xsl:template as well as the file name. Two weblets with the same xsl:template name cannot be used together on the same layout or webroutine.

## 2.2 Technology Services

Before you launch into any WAM development it is important to understand what Technology Services are and how they impact your WAM development.

### What is a Technology Service?

Technology Services apply to the Presentation Layer or Web Design of a WAM. They allow common business logic (the RDMLX) to render a presentation on multiple types of client. This is an important concept as it allows the RDMLX of a single WAM to be presented in multiple technology formats, on multiple devices, separating application logic from the presentation technology.

So essentially a Technology Service defines the presentation output of a WEBROUTINE.

### Which Technology Service should I use?

It depends on how you want to deliver your web solution.

Two of the most commonly used Technology Services are shipped with Visual LANSA to support the presentation layer for WAMs. These are XHTML (eXtensible Hypertext Markup Language) to support a standard web browser interface and JQMOBILE (jQuery Mobile) designed for mobile devices.

If you want to present your web application in a format other than XHTML or JQMOBILE, you will need to create your own Technology Service.

### Can I create my own Technology Services?

Yes, you can define additional Technology Services but the implementation and resulting XSL are entirely your responsibility.

Technology Services are defined in the LANSA Editor. To create a new Technology Services, use the New button from the toolbar and select Technology Services. Then fill in the details for your new Technology Service.

You will need to create your own XSL Stylesheet template documents to support your new Technology Service. These documents should be saved in the appropriate LANSA directory ...\\X\_WIN95\\X\_LANSA\\web\\tsp. These templates will be used to generate the initial XSL presentation of a WEBROUTINE.

Refer to [Technology Services](#) for further information about creating your own Technology Service.

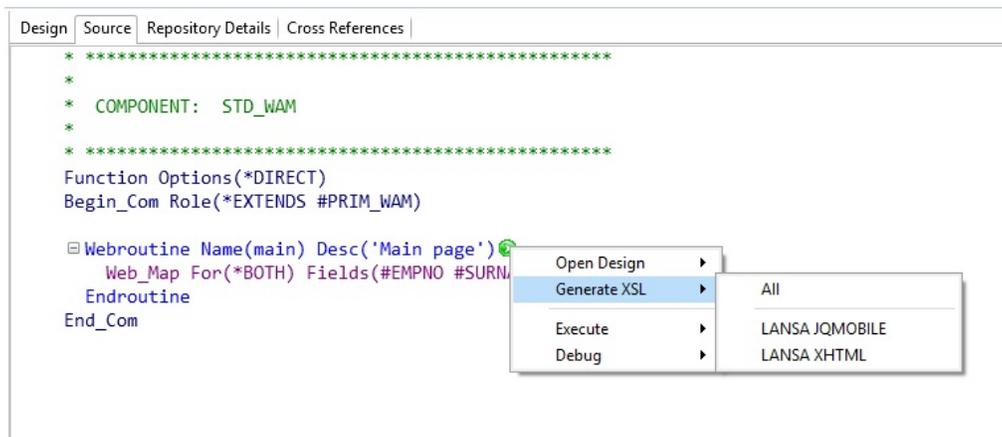


Do not add additional Technology Services to the LANSAs provider as the LANSAs provided Technology Services may be extended or changed in future versions.

## How do I generate XSL for a particular Technology Service?

When you create a new webroutine you get the option to generate XSL for existing Technology Services. If you only intend to execute your finished web application on a web browser you would not select the JQMOBILE Technology Service.

If you want to generate XSL for a different Technology Service at a later stage you can do so by right mouse clicking on the Webroutine Design Glyph, choosing Generate XSL and then the Technology Service Provider you are interested in.



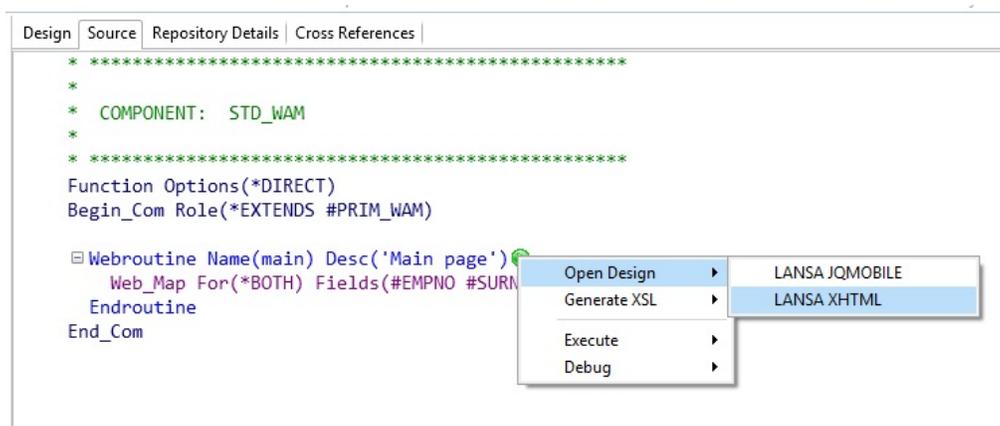
## How do I view the presentation for a specific Technology Service?

The default Technology Service for the LANSAs Editor is XHTML. Use the Web menu in the LANSAs Editor to view the JQMOBILE design (if it was generated) or any other Technology Services you have created and subsequently generated Web Designs for.

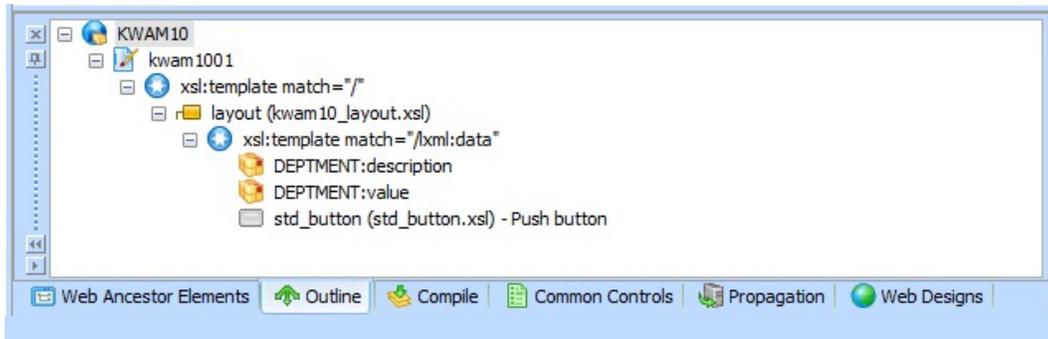
## 2.3 Structure of a Webroutine's XSL

We have examined what objects are created when you compile a WAM and the role of Technology Services with these objects. Now let's delve further into the workings of the WAM by opening the XSL generated for a webroutine in the LANSa Editor.

Using the same WAM definition (*KWAM10*), you can open the *KWAM1001* Webroutine XSL object (that is, *kwam10.kwam1001.xml*) in the LANSa Editor's Design tab by clicking on the  Webroutine Design Glyph or by right mouse clicking , choosing *Open Design* and then the *Technology Service Provider* you are interested in..

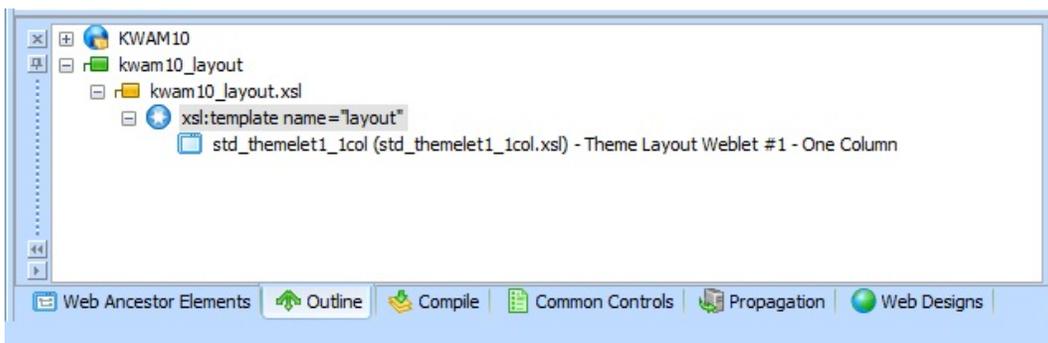


Now that the Web Design is open in the LANSa Editor, select the *Outline* tab. You may need to explicitly open it from the LANSa Editor's *View* menu, choosing *Views* and then *Outline* or by pressing F6. You will see a tree view representing the various weblets, HTML, XML and XSL structures that make up the presentation. A prominent feature high in the tree view is the reference to the automatically generated WAM layout *kwam10\_layout.xml*. There is also a field description and value included in the XSL (but are not represented as weblets) for the mapped field *DEPARTMENT*. In addition, the XSL contains the weblet *std\_button* which is the display for a push button.



If you double click on the WAM layout weblet in the *Outline* tab (or press right click and choose *Weblet: kwam10\_layout - Open* from the context menu), it will open the **kwam10\_layout.xml** weblet in the LANSAXML Editor as well. Looking at the Outline tree structure of this WAM layout shows references to more weblets, in this case **std\_themelet1\_1col**. We begin to understand how weblets are reused, even in the automatically generated XSL objects.

Back to our WAM layout weblet, looking at the Outline tree you can quite simply deduce from this that the WAM layout is based on the standard one column themed layout weblet named **std\_themelet1\_1col**.



You can verify this in the Design tab:



Also notice that the WAM layout weblet has no references to WebRoutine-specific details (that is, the mapped webroutine information). It is the shell that provides structure and a consistent interface for the page.

Note that for the purpose of this introduction to the structure of a Webroutine's XSL, we are using an example based on the `std_themelet1_1col` weblet.

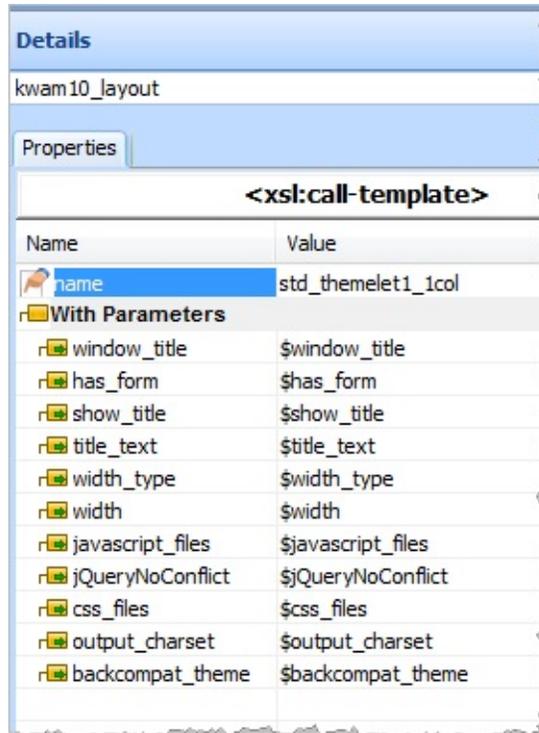
Because the shipped layout weblets (prefixed by `std_`) must never be modified, it is recommended that you always build your own site layout weblet. The site layout weblet can, of course, be modified as you require for your web site. The easiest way to create your own site layout is to use the [Web Application Layout Manager Wizard](#).

If you were using your own site layout weblet named `kwamsite`, the Outline tree would look like the following:

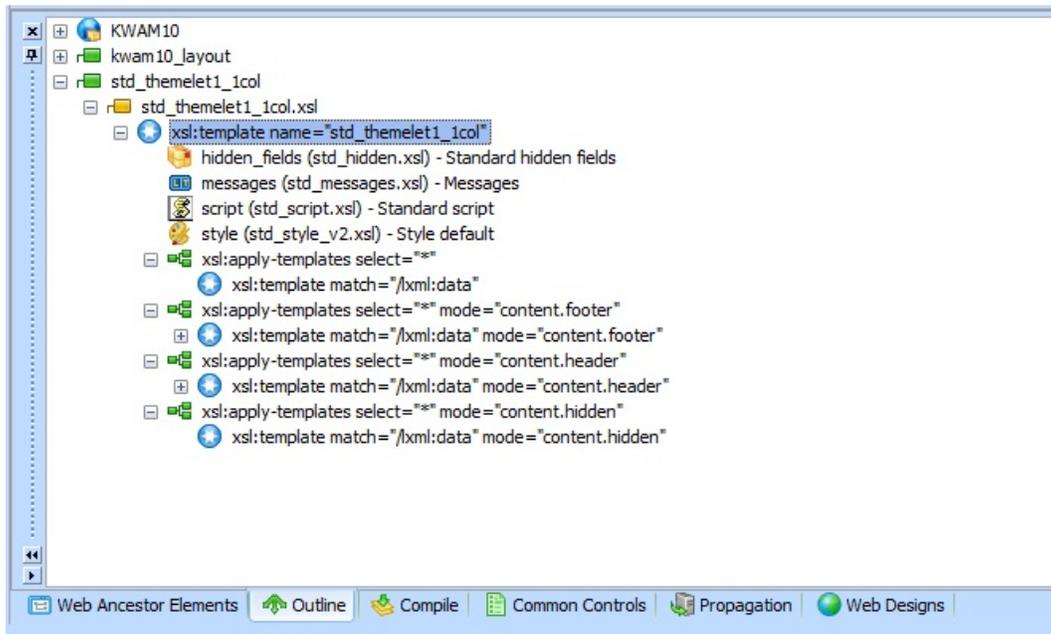


In the following text, the `std_themelet1_1col` weblet name would be replaced by the `kwamsite` weblet name when using a site layout. All other details would be the same if the **Web Application Layout Manager Wizard** was used to generate this site layout.

Still looking at the WAM layout weblet, if you click on the *Details* tab, the property settings for the weblet `kwam10_layout.xsl` are exposed. These are currently set to the default values. Changing these properties will change the layout interface accordingly.



At this point, if you move back to the *Outline* tab, you can continue to drill down through the weblets to further investigate how the basic webroutine kwam1001 is constructed. From the WAM Layout weblet you can double click on the std\_themelet1\_1col weblet so it too opens in the LANSa Editor. Note that you must not update this shipped standard weblet.

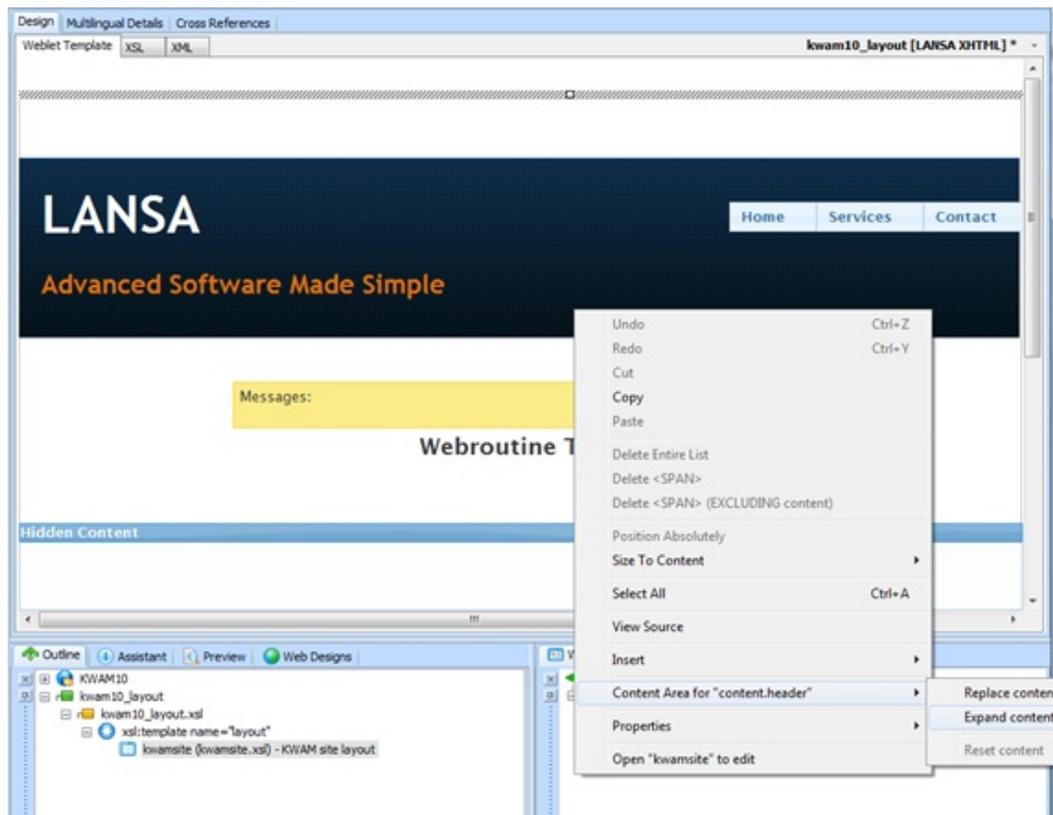


Looking at the structure of the std\_themelet1\_1col weblet in the Outline tab (or

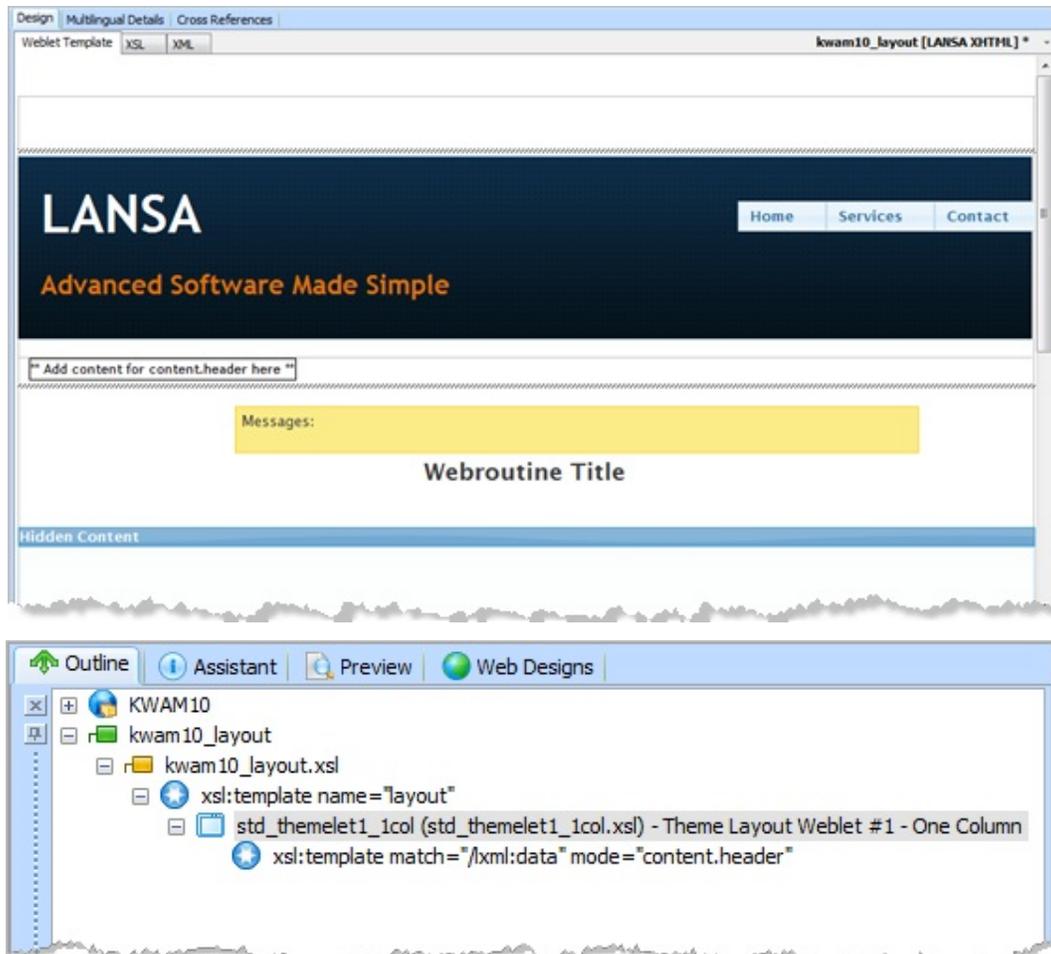
the *Design* view) you will see references to page content areas identified by the template names **content.header**, **content.hidden** and **content.footer**. Even though the kwam10\_layout does not specifically show these names in the *Outline* view, they are available as editable areas of both the kwam10\_layout and the kwam1001 webroutine Web Designs using the LANSA editor.

- Review the std\_themelet1\_1col weblet in the Design view to observe the relationship between this weblet and the WAM layout kwam10\_layout.
- Close the std\_themelet1\_1col weblet without updating to return to editing the kwam10\_layout Weblet for the WAM.

To update any of the page content areas, position the cursor to the content area to be modified (header, footer or hidden) and press right click. The context menu option Content Area for the applicable page content will be available to allow you to Replace or Expand the content:



If the content area is modified, it will then appear in the outline view for the WAM layout kwam10\_layout. For example:



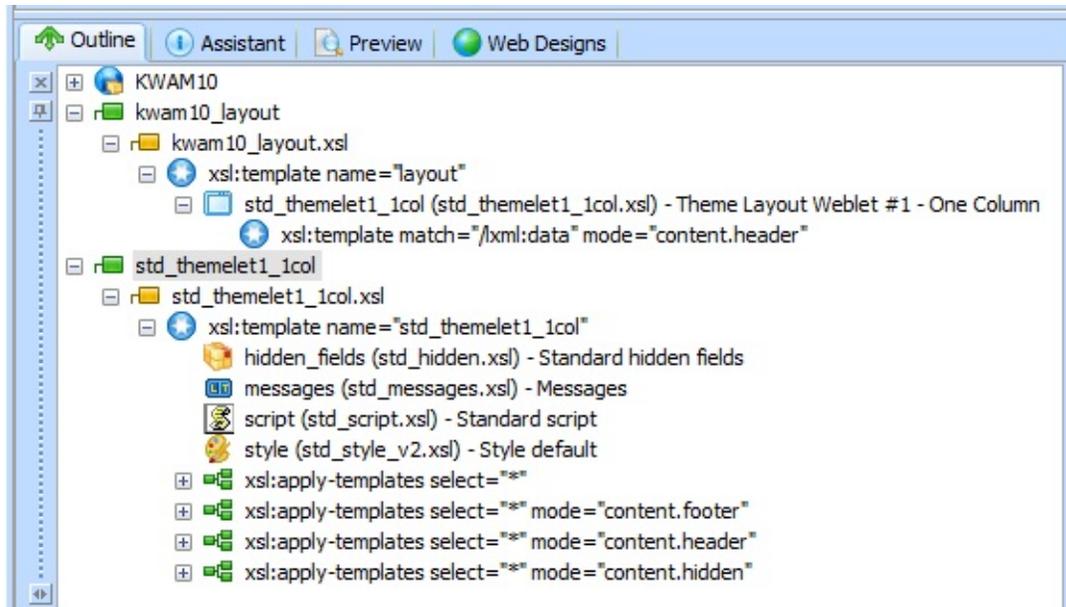
Now let's look in more detail how the Webroutine layout **kwam1001\_layout** is constructed from each of its related weblets.

First, open the XSL tab for the **kwam1001\_layout** weblet and scroll to the top of the XSL. You don't need to understand the code but observe the references to "import" other XSL documents:

```
<xsl:import href="std_themelet1_1col.xsl" />
<xsl:import href="std_types.xsl" />
```

These imports indicate that the **kwam1001\_layout** weblet refers to a set of weblets (that is, XSL documents), which, although they may not always be visualized, are important elements in the definition.

From the *Outline* view of the **kwam10\_layout** WAM Layout weblet, double click on the **std\_themelet1\_1col** weblet so it also opens in the LANSA Editor. Review the updated *Outline* view expanded to include the **std\_themelet1\_1col** weblet.



Again, open the XSL tab for the **std\_themelet1\_1col** weblet, scroll to the top of the XSL and observe the references to "import" other XSL documents:

```

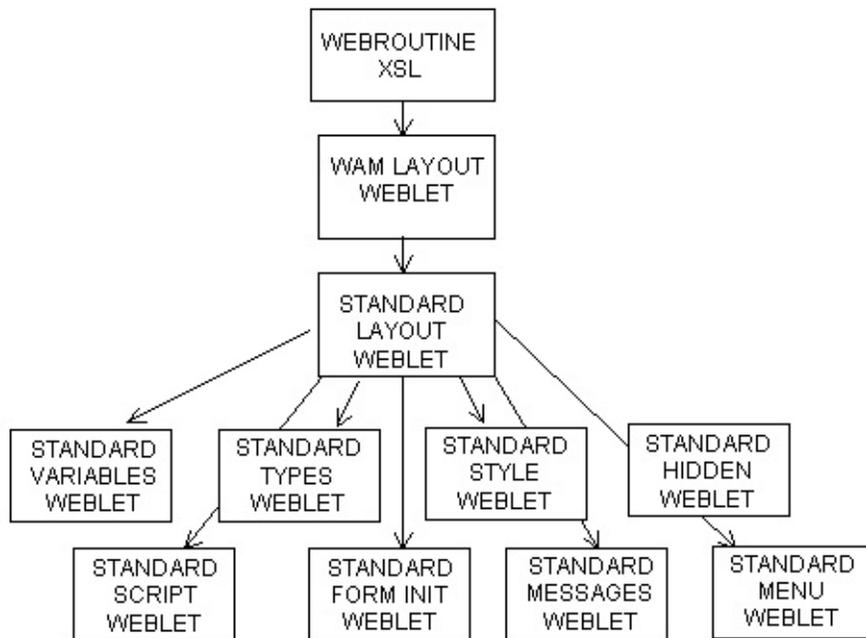
<xsl:import href="std_variables.xsl" />
<xsl:import href="std_types.xsl" />
<xsl:import href="std_hidden.xsl" />
<xsl:import href="std_style_v2.xsl" />
<xsl:import href="std_script.xsl" />
<xsl:import href="std_menubar.xsl" />
<xsl:import href="std_messages.xsl" />

```

These imports indicate that the **std\_themelet1\_1col** weblet refers to another set of weblets (that is, XSL documents), which, although they may not always be visualized, are important elements in the definition.

For now, all you need know is that these different weblets are referred to in the shipped standard layouts and as such are "available" to any other weblets, which in turn refer to these layouts. We will describe what these various weblets do in the following sections of this document.

The following diagram summarizes what we have just described about the structure of a webroutine's generated XSL:



You will find more details of each of these pieces elsewhere in this guide.

## **2.4 WAM Layouts and Layout Weblets**

In previous descriptions we have touched upon the concepts of WAM layouts and Layout weblets. Now it is time to explain:

[2.4.1 What is a WAM Layout?](#)

[2.4.2 What is a Layout Weblet?](#)

[2.4.3 What do Layouts Determine/Control?](#)

[2.4.4 How is a WAM layout assigned to a WAM?](#)

[2.4.5 How do I Create my Own Site Layout?](#)

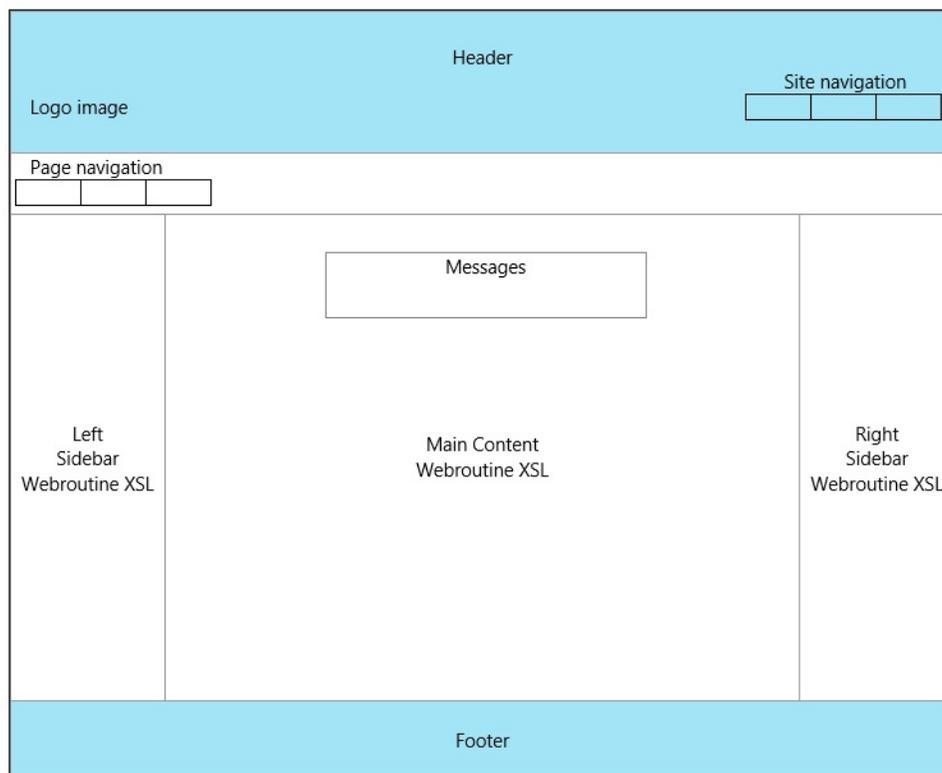
[2.4.6 Can I Change the WAM Layout used by a Webroutine?](#)

[2.4.7 Can I Change the Layout Weblet associated with a WAM Layout?](#)

You can easily create your layout using the Web Application Layout management Wizard [Web Application Layout Manager Wizard](#).

## 2.4.1 What is a WAM Layout?

A WAM layout weblet is a specific type of weblet that is used to give structure to the web page. Typically, it will define any titles, menus, message presentation or logos to be displayed. The WAM layout also controls the Cascading Style Sheet or other documents to be applied. (We'll get to this later in [2.7.2 What CSS files are loaded and how do I add my own?](#))



The WAM layout weblet is used as the basis for any presentation associated with the WAM's webrouines.

A single WAM layout is generated for each WAM regardless of how many webrouines are defined within the WAM. If your web application includes multiple WAMs, the same layout can be applied to all the WAMs in your application. This way, you can guarantee a consistent interface.

Contrary to what the name suggests, a WAM layout does not have to be made of visual elements – although it usually is.

The non-visual elements of a layout include references to XSL documents for:

Standard variables

Standard data types

Style

JavaScript

Default hidden fields

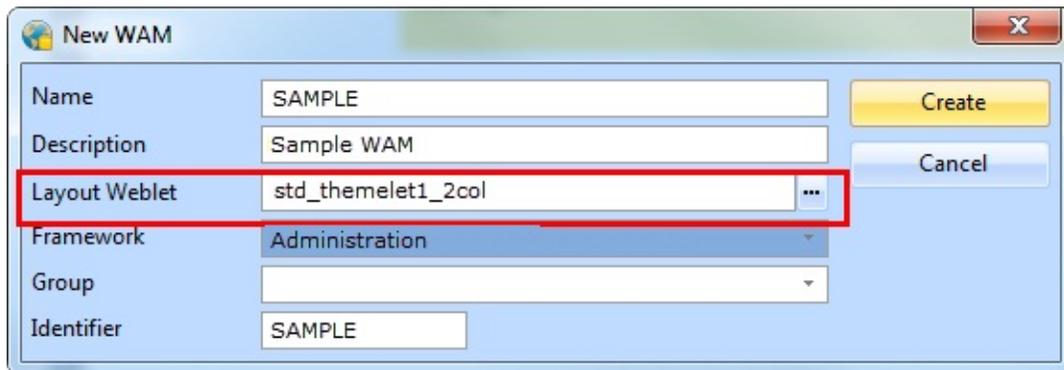
## 2.4.2 What is a Layout Weblet?

If you have many WAM layouts and want them to share common elements, you can place the common elements in a Layout Weblet. Similar to other weblets, the purpose of the Layout Weblet is to reuse functionality and avoid unnecessary duplication. Refer to [Create a Weblet](#) in the *Visual LANSA User Guide* for details about Weblets.

Three themable Layout Weblets (Themelets) are shipped with Visual LANSA (std\_themelet1\_[1-3]col). These layouts can be used for any WAM-specific layouts or as a starting point for creating your own site Layout Weblets.

The easiest way to create your own site layouts is to use the [Web Application Layout Manager Wizard](#).

When you create a new WAM you can select the site's Layout Weblet. The WAM layout that is automatically created is based on the Layout you nominate.



The screenshot shows a dialog box titled "New WAM" with a close button (X) in the top right corner. The dialog contains several input fields and buttons:

- Name:** Text box containing "SAMPLE".
- Description:** Text box containing "Sample WAM".
- Layout Weblet:** Text box containing "std\_themelet1\_2col" with a red rectangular highlight around it. To the right of the text box is a small "..." button.
- Framework:** A dropdown menu currently showing "Administration".
- Group:** An empty dropdown menu.
- Identifier:** Text box containing "SAMPLE".
- Buttons:** A yellow "Create" button and a grey "Cancel" button are located on the right side of the dialog.

### **2.4.3 What do Layouts Determine/Control?**

The layout (A standalone WAM layout or a Layout weblet based WAM layout) is a key element in the generated webroutine presentation. It ensures that a consistent interface is available across webroutines.

When you view your webroutine in the LANSA Editor's Outline tab, the layout weblet is generally at the highest level in the outline tree. This indicates that all weblets below the layout in the tree can refer to the documents specified in the layout weblet.

Some of the things controlled by layouts include:

- The appearance of any menus
- Available menu options
- The appearance of a message box
- The Cascading Style Sheet to be applied
- Access to common JavaScript functions
- Definition of any global hidden fields
- Standard variable definitions which may be referenced in other weblets
- Whether a visual layout should be applied.

Of course if you define your own layout, you can decide what common elements need to be included in the interface.

#### **2.4.4 How is a WAM layout assigned to a WAM?**

A WAM-specific layout weblet is automatically generated for a WAM the first time it is built or compiled unless one already exists.

By default, when XSL is generated, the processor checks if a WAM-specific layout weblet already exists for the WAM. If a WAM layout does not exist, a new WAM layout weblet is generated and stored in the repository where the name is composed of the WAM name followed by "\_layout". After it has been generated, your WAM-specific layout weblet is referenced by all the webrouines in the associated WAM. Any changes to the WAM-specific layout weblet will be reflected in all of the WAM's webrouines.

The *Generate XSL* options on a WAM compilation do not regenerate the WAM-specific layout. A WAM-specific layout is generated only once. Any subsequent modifications to the WAM-specific layout, or the assignment of a different layout, must be performed in the LANSAs Editor.

## 2.4.5 How do I Create my Own Site Layout?

The easiest way is to run the [Web Application Layout Manager Wizard](#).

To create a layout weblet from scratch, select the Weblet option in the LANSAs Editor's New toolbar dropdown list. In the *New Weblet* dialog select the Layout Weblet option to create a Layout weblet.

Alternatively, you can use any of the shipped layout weblets as a basis for creating your own.

## **2.4.6 Can I Change the WAM Layout used by a Webroutine?**

In the WAM Editor's Designer tab, drag and drop the WAM Layout you want to use. It will replace the existing WAM Layout.

## 2.4.7 Can I Change the Layout Weblet associated with a WAM Layout?

After a WAM layout weblet has been created for a WAM, you can open and edit this weblet in much the same as any other weblet. You can modify the properties associated with the WAM layout or you can replace the Layout weblet it is based on, using an alternative Layout weblet as the base.

To change the layout weblet used as a template for the WAM layout:

1. Open the WAM layout weblet in the LANSAs Editor. Select the Design tab.
2. In the Repository tab, locate the weblet to be used as a layout. Drag this new layout weblet onto the WAM-specific layout weblet. The new layout will be displayed below the old layout.
3. Select the old layout and delete it.
4. Save the changes.



It is important to execute the above steps in the prescribed order. By adding a new layout weblet to be used as the template, all existing objects are moved under this new template. After this step is complete, you can safely remove the old layout template without removing the dependant objects.

## 2.5 Variables

The variables weblet – `std_variables` – is not visualized.

The sole purpose of the `std_variables` weblet is to define a set of variables with default values. These default values can in turn be referenced in appropriate weblet properties or used in other weblet's XSL source. For variables defined in `std_variables` to be referenced by other weblets, the referencing weblet must include a specific import of the `std_variables.xsl` document.

For example, the `std_menu_item` weblet includes the following statement in the XSL source:

```
<xsl:import href="std_variables.xsl" />
```

Further down in the XSL is a reference to a variable `$lweb_WAMName` (defined in `std_variables`) to resolve the name of the currently executing WAM at runtime.

## 2.5.1 How can I Change the Value of a Shipped Variable?

Variables in XSL are not like the variables you have come across in other programming environments. In XSL, the term variable is used in its mathematical sense to mean a placeholder. This means that, once defined, a variable cannot be changed.

It is possible to override a variable in certain contexts. To do this correctly you need to have a good understanding of XSLT import precedence. The variables defined in `std_variables` provide access to environment parameters and specific parts of the source `lxml` data. If you need to change the default value of a variable then change the source data rather than the variable.

If you wish to create your own variables then create your own `xxx_variables` weblet and import it into your layouts or weblets as required.

We do not recommend that you change the `std_variables` weblet as this may be changed by future updates.



If you reinstall Visual LANSA, any changes made to the standard shipped weblets will be overwritten.

## **2.5.2 How can I Create my own Variables?**

The safest way to create your own variables is to create your own `xxx_variables` weblet and import this wherever you need access to the new variables.

You can use the existing `std_variables` weblet as a guide but don't attempt to redefine variables already defined in `std_variables`. This is not necessary and may result in strange and unexpected behavior.

Some knowledge of XSLT, and a thorough understanding of where and how variables are used, is required before attempting to create your own variables.

## 2.6 Localized Variables

The locale definitions weblet – `std_locale` – is not visualized.

The purpose of the `std_locale` weblet is to define a set of variables with default values that may differ for different locales (regional settings). These default values can in turn be referenced in appropriate weblet properties or used in other weblet's XSL source. For variables defined in `std_locale` to be referenced by other weblets, the referencing weblet must import `std_variables.xsl` which itself imports `std_locale.xsl`.

This weblet is shipped in different languages with variables set to locale specific values.

For example, the `std_style_v2` weblet refers to the variable `$lweb_std_css_language_overlay` to obtain the name of a language specific style sheet to be applied to the layout.

You can create your customized version of localizable variables, for example, if you want to customize variable `$lweb_std_css_language_overlay`.

## **2.7 Cascading Style Sheets (CSS) and the Style Weblet**

The `std_style` weblet adds the CSS stylesheets needed by the WAM, any additional CSS stylesheets you nominate (indirectly via your layout) and external resources CSS stylesheets used by your WAM.

Together, the `std_style` weblet and external Cascading Style Sheet definitions can be used to tailor the overall appearance of your web page interface.

Let's start with the basics:

[2.7.1 What are Cascading Style Sheets and how do they work?](#)

[2.7.2 What CSS files are loaded and how do I add my own?](#)

[2.7.3 Can I create my own Style Weblet?](#)

[2.7.4 What Cascading Style Sheets are available?](#)

## 2.7.1 What are Cascading Style Sheets and how do they work?

A Cascading Style Sheet tells the browser how to display page elements. Cascading Style Sheet information determines things like the fonts and color schemes, visual effects, alignment, border size and color, but may also be used to define images and other features related to the interface. These properties can be assigned to individual elements identified by an ID, or groups of elements identified by type, location and class.

A detailed description of CSS is beyond the scope of this document. However, there are many good books and online resources that cover the subject in detail. A good place to start online is the free tutorial at [W3Schools](#).

Many of the shipped weblets include style (or class) properties. The default style applied to a property, and the full set of styles available in the dropdown list associated with these properties, relate directly back to the CSS file referenced on the WAM's related layout.



Cascading styleheets are shipped minified (Most whitespace removed). Non-minified version of these files are also shipped in the same directory.

LANSa also ships with a set of CSS files defined specifically for use with the PocketPC Technology Service.

## 2.7.2 What CSS files are loaded and how do I add my own?

The `std_style_v2` weblet takes care of creating all the `<link>` tags needed to load the CSS files so you need to include it in the `<head>` section of your layouts.

The `std_style_v2` weblet always loads `std_style.min.css` into every layout. This defines the non-theme related properties of all LANSA supplied weblets.

It then loads any CSS files defined by its `theme_css_filename` and `css_files` properties. These properties are provided for backwards compatibility with layouts built with older versions of the weblet. For new layouts, you should specify 'none' in `theme_css_filename` and use External Resources to define additional CSS files you want to include.

Next, it adds any CSS files defined as External Resources (which we mentioned above) referenced in the webroutine, layout or weblets used by the webroutine.

Finally, the `std_style_v2` weblet loads a stylesheet defined by the variable `$lweb_std_css_language_overlay`. This variable is defined in the [std\\_locale](#) weblet and provides a means to apply language specific CSS modifications.

 Before External Resources and jQuery UI support, the default LANSA theme was included in `std_styles.css` and always loaded. Any custom themes added with the `theme_css_filename` property had to take this into account and undo any styles from the default theme that they didn't want. From **LANSA version 12 SP1** the styles associated with this default theme are separated into their own CSS file (`theme_default.css`). For backwards compatibility this theme is still loaded before any theme specified in `theme_css_filename`.

New themes can remove `theme_default.css` and start with a blank canvas by specifying 'none' in the `theme_css_filename` property.

The `css_files` property provided a mechanism for individual webroutines to add special purpose CSS files to the page such as a CSS file needed by a custom weblet. External Resources are a much more powerful mechanism for doing this, allowing weblets to define their own CSS requirements and having that automatically communicated through to `std_style_v2`. The `css_files` property is no longer necessary and is included only for backwards compatibility.

**Notes:**

The CSS filenames passed in arguments to `std_style_v2` are assumed to be relative to the style sub-directory. The value of the style sub-directory is defined by the variable `$lweb_style_path`. By default, this variable refers to the sub-directory `/style` directly under the web server image directory. We recommend that you do not change this value.

**From version 13.0**, WAM output is in UTF-8.

`<link>` elements for the `ccs_files` property didn't have a `charset` attribute, therefore they were assumed to be in the same character set as the main document. To preserve backwards compatibility, they now have a `charset` attribute that defaults to `"shift_jis"` for language JPN and to `"iso-8859-1"` for all other languages. If you need to nominate a different character set, use the `ccs_files_charset` property in weblet `std_style_v2` (you will need to add this parameter to your site layout). A better approach is to register your extra CSS files as external resources and include them as such.

### 2.7.3 Can I create my own Style Weblet?

The standard style weblet should handle most of your needs but, if you want to create your own style weblet, you can. The standard style weblet provides functionality that is essential to the correct operation of the LANSAs supplied weblets and layouts. You should always call it from within your custom style weblet or design your weblet, as shown in the example below, to work alongside the standard style weblet.

```
<xsl:import href="std_style_v2.xsl" />

<xsl:template name="my_style">
  <xsl:call-template name="style">
    <xsl:with-param name="theme_css_filename"/>
    <xsl:with-param name="css_files"/>
  </xsl:call-template>

  <!-- Custom style functionality here -->
</xsl:template>
```

You **must not** use a template name of "style".

If you name the template of your custom weblet "style" then you will cause an infinite loop.

## 2.7.4 What Cascading Style Sheets are available?

The main CSS stylesheets mentioned in [2.7.1 What are Cascading Style Sheets and how do they work?](#) are in the main style directory under the images directory.

The themed CCS stylesheets are under the jQuery subdirectory—under a subdirectory named after the theme.

See [External Resources Shipped with LANSA](#) for a list of the other stylesheets available to WAMs.

## 2.8 JavaScript and the Script Weblet

### 2.8.1 Can I create my own Script Weblet?

### 2.8.2 How do I Format inline JavaScript for a Weblet Property?

The script weblet – `std_script` – is not visualized.

The `std_script` weblet loads a number of external JavaScript files and initializes a number of JavaScript variables and functions used by the LANSA weblets. It should be included in the `<head>` section of all layouts.

The external JavaScript files referenced are a small set of JavaScript files installed on the web server to support WAMs. These scripts are loaded into a subdirectory `/script` directly under the image directory. If you create your own script weblet, ensure that the scripts included in the shipped `std_script` weblet are included in your script weblet.



These files are shipped minified (Most whitespace removed). The non-minified versions of these files are also shipped in the same directory.

If you wish to provide your own localized versions of the JavaScript messages, make a copy of `std_script_messages.min.js` and translate the messages. Then edit the language specific version of the `std_locale` weblet and update the `lweb_script_messages_file` and `lweb_script_messages_file_charset` variables.

To add your own JavaScript files to a page or layout, Enroll your JavaScript file as a Web Images external resource. Add the external resource to the webroutine or weblet that requires it. The corresponding `<script>` element will be added at runtime automatically.

#### Notes:

- Every layout has a `javascript_files` property, which is passed to the `std_script` weblet. This parameter takes a comma-delimited list of file names (assumed to be relative to the `/script` directory).
- From **Version 13.0**, WAM output is in UTF-8. `<script>` elements for the `javascript_files` property didn't have a `charset` attribute, therefore they were assumed to be in the same character set as the main document. To preserve backwards compatibility, they now have a `charset` attribute that defaults to `"shift_jis"` for language JPN and to `"iso-8859-1"` for all other languages. If you need to nominate a different character set, use the

javascript\_files\_charset property in weblet\_std\_script (you will need to add this parameter to your site layout). A better approach is to register your extra JavaScript files as external resources and include them as such. Several of the shipped weblets also include small inline JavaScript functions in the XSL.

## 2.8.1 Can I create my own Script Weblet?

The standard script weblet should handle most of your needs but, if you want to create your own script weblet, you can. The standard script weblet provides functionality that is essential to the correct operation of the LANSAs supplied weblets and layouts. You should always call it from within your custom weblet or design your weblet, as shown in the example, to work alongside the standard script weblet.

```
<xsl:import href="std_script.xsl" />

<xsl:template name="my_script">
  <xsl:call-template name="script">
    <xsl:with-param name="javascript_files"/>
    <xsl:with-param name="trap_script_errors"/>
  </xsl:call-template>

  <!-- Custom script functionality here -->
</xsl:template>
```

You **must not** use a template name of "script".

If you name the template of your custom weblet "script" then you will cause an infinite loop.

## 2.8.2 How do I Format inline JavaScript for a Weblet Property?

Any weblet property where JavaScript is a valid or expected value will accept an inline JavaScript enclosed in single quotes. The format of the JavaScript is your responsibility.

All inline JavaScript must end with a semicolon (;) For example, 'alert("hello world");'. Note that, because the JavaScript must be enclosed in single quotes (this is done for you in the background), you cannot use a single quote within the code. If you need to do this, consider creating a function in an external JavaScript file and calling it from your inline code.

Most properties that expect JavaScript are executed in response to an event or just prior to performing some action. For example, the `presubmit_js` property of the `std_button` weblet is executed just before the form is submitted to the server. This gives you the opportunity to provide some extra processing or to cancel the event/action. To cancel the event/action you must use "return false;"

For example:

**if ( confirmWithUser() == false) return false;**

(where `confirmWithUser` is a function you have defined in an external JavaScript file)



### JavaScript Notes:

1: Do not use `return` or `return true` in your inline JavaScript. This has a similar effect to `return false` in that it stops execution of the LANSAs JavaScript but it does not stop the browser from performing its default event handling. This may result in strange and unexpected behavior.

2: The `{` and `}` characters have a special meaning in XSLT and cannot be used in a JavaScript property. Doing so will cause strange behavior. If you need to write more complex JavaScript that requires these characters you should create a separate JavaScript function and call it from your property.

3: Previous versions of the documentation advised ending the JavaScript with a double backslash (`//`) to cancel the default processing. This technique has the same effect as using `return` and should not be used.

## 2.9 Messages

The messages weblet – `std_messages` – formats the presentation of any application messages on a web page.

Message presentation is automatically incorporated at the top of all the standard shipped layouts which means you generally do not have to review, modify or even apply the `std_messages` weblet to your presentation.



A `show_messages` property is included on each shipped layout to indicate whether the message box generated by `std_messages` should be incorporated into the layout. The default setting for this property is to show messages. If you do not want to show messages or choose to display your messages in a different window, this property should be changed to false.

You can modify the basic appearance such as background color, of the message box interface by redefining the appropriate style sheet classes in your own CSS file. Alternately if you want a completely different visualization of the messages you will need to create your own version of the `std_messages` weblet and refer to this as required in your presentation interface.

## 2.10 Types

The type weblet – std\_types – is not visualized.

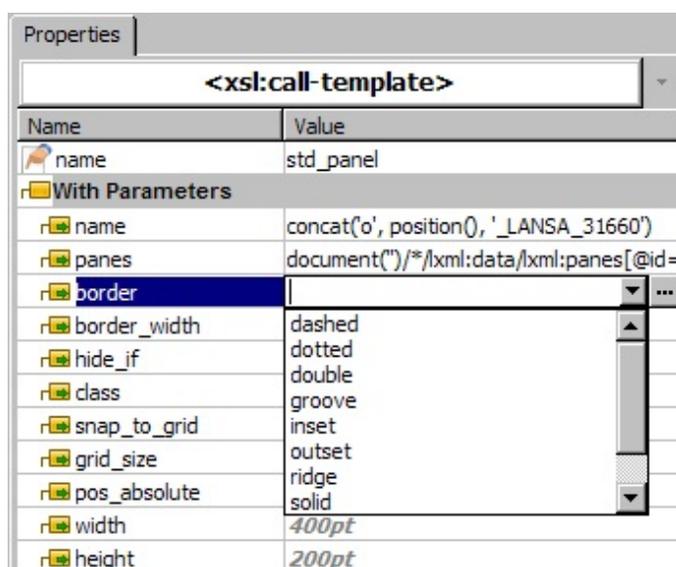
The std\_types weblet defines the type of information that is valid to be entered for a weblet property. Types are declared in the XSL source by providing an attribute wd:type, assigning a type name and then detailing the type of information which is valid for this type.

For example, locate the following code in the std\_types document:

```
<wd:type name="std:border_style">
  <wd:enumeration value="'dashed'" />
  <wd:enumeration value="'dotted'" />
  <wd:enumeration value="'double'" />
  <wd:enumeration value="'groove'" />
  <wd:enumeration value="'inset'" />
  <wd:enumeration value="'outset'" />
  <wd:enumeration value="'ridge'" />
  <wd:enumeration value="'solid'" />
  <wd:enumeration value="'window-inset'" />
</wd:type>
```

It is easy to deduce from this type definition that any weblet property that refers to this type std:border\_style will relate to a border style and include a valid set of values which correspond to the values on the wd:enumeration statements.

To verify this, add a panel (std\_panel) weblet to a web page and select the  Details tab to review the associated panel properties. Check the values available in the dropdown list associated with the border property. As you would expect the dropdown's values match the wd:enumeration statements.



Now open the std\_panel weblet and review the XSL source to see how the

relationship between a weblet property and the type is established.

First of all note that the `std_types` document is imported into the `std_panel` weblet's XSL source:

```
<xsl:import href="std_types.xsl" />
```

Now scan down the XSL source and you will find a relationship defined between the weblet property and the type definition.

```
<xsl:template name="std_panel">
  ...
  <xsl:param name="border" wd:type="std:border_style" />
```

So it all comes together!



Tip: If you are defining your own weblets, you may refer to the types defined in the `std_types` document to indicate what values are valid for your weblet properties. It is not anticipated that you will need to create your own types.

## 2.11 Hidden

The hidden fields weblet – std\_hidden – is not visualized.

This weblet provides access to a group of internally defined and evaluated properties. The values assigned to the respective properties are determined in the RDMLX definition and are required to execute a webroutine in a web browser.

The standard hidden information is:

SERVICENAME  
WEBAPP  
WEBROUTINE  
PARTITION  
LANGUAGE  
SESSIONKEY  
LW3TRCID

The hidden information relates directly to the values that can be included on a URL to invoke a webroutine. For example you may execute a webroutine MaintainRecord in WAM WAMEX02 directly from a web browser using the URL:

**http://<server>/cgi-bin/lansaweb?  
wam=WAMEX02&webtrn=MaintainRecord&part=DEX**

or alternately, assuming the servicename WAMEX02 Maintenance has been assigned to the webroutine, the URL may look like this:

**http://<server>/cgi-bin/lansaweb?  
srve=WAMEX02\_Maintenance&part=DEX**

In both examples the language, tracing and any other undeclared parameter values will assume the default value.

## **2.12 Keys**

The keys weblet – `std_keys` – is not visualized.

The standard keys weblet declares a set of named keys, which can be used in other XSL documents to allow easy access to complex XML documents. Use of key information requires a thorough understanding of XSL and XML.

This weblet is imported into many weblets to support use of the key function in these weblets.

### **Further Reading**

For further information about WAMs and their use refer to: [Weblets and Weblet Templates](#).

## 2.13 Inline Lists

### Rationale

Weblet XSL templates are heavily parameterized in order to give the WAM developer the ability to customize the results.

For example, the `std_anchor` weblet has the ability to change its appearance when the mouse hovers over it. To activate this behavior, the developer assigns a value to the `mouseover_class` property. The following XSLT is executed at runtime to see if the property has been set and, if so, adds `onmouseover/onmouseout` event handlers to the anchor:

```
<xsl:if test="$mouseover_class != "">
  <xsl:attribute name="onmouseover">
    <xsl:text>this.className='</xsl:text>
    <xsl:value-of select="$mouseover_class" />
    <xsl:text>'</xsl:text>
  </xsl:attribute>
  <xsl:attribute name="onmouseout">
    <xsl:text>this.className='</xsl:text>
    <xsl:value-of select="$class" />
    <xsl:text>'</xsl:text>
  </xsl:attribute>
</xsl:if>
```

Once the WEBROUTINE design has been saved, the value of `$mouseover_class` never changes but this code is still executed every time the webroutine is run. If the weblet is in a list, the code is executed again for every row of the list.

Now, in most cases, many of the parameters (weblet properties) are constant and don't depend on runtime values. It is more efficient to apply these properties once at design time instead of doing it every time at runtime. This is particularly important in the case of lists (even more so for large lists).

This is what inline lists do. Inline lists differ from standard lists in that the XSL is done at design time. All weblet properties that can be applied at design time are resolved, and special extension elements and functions are used to allow WAMs to use runtime values where needed.

Applying the XSL during design time means that you can't customize an inline list with information (such as field values) that is only available at runtime. This is a trade-off. If you need this extra flexibility you use a standard list. If you don't need it you can benefit from the better performance provided by an inline list.

## 2.13.1 Creating an Inline List

There are two ways to make your list an inline list.

1. Setting the *Inline* property of your WAM to *Lists* will make all lists in the WAM inlined by default.

```
FUNCTION OPTIONS(*DIRECT)
BEGIN_COM ROLE(*EXTENDS #PRIM_WAM) INLINE(Lists)
```

```
END_COM
```

2. Marking an individual list as inline in the WEB\_MAP.

```
WEBROUTINE NAME(MyWebroutine) DESC('Sample Webroutine')
  WEB_MAP FOR(*OUTPUT) FIELDS( (#DEPTLIST *INLINE) )
```

```
ENDROUTINE
```

If any XSL already exists when you change a list to or from inline, the XSL will need to be regenerated or the list deleted and re-added to the XSL.

If inline lists are turned on at the WAM level, an individual list can be marked as not inline in the WEB\_MAP like this:

```
WEB_MAP FOR(*OUTPUT) FIELDS( (#DEPTLIST
*NOINLINE) )
```

## 2.13.2 Using Weblets in an Inline List

Weblets must be "inline-aware" before they can be used in an inline list. An alert will notify you if you attempt to drop a weblet that is not inline-aware onto an inline list.



For the most part, inline weblets look and behave the same as their non-inline counterparts. The key difference is that the inline weblet is generated when you drop it onto a list and, again, each time you change a property. Because of this, there are a few considerations that you need to be aware of.

### Accessing field and column values

In a non-inline list, you would access the value of a column with the `$COLUMNNAME` XSL variable or a field value with an XPath expression such as `key('field-value', 'FIELDNAME')`. Because the XSL in an inline weblet is executed at design time, no XSL like this cannot be used to access runtime data. There are several special XSL extension functions you can use to access runtime data in a weblet property. These are:

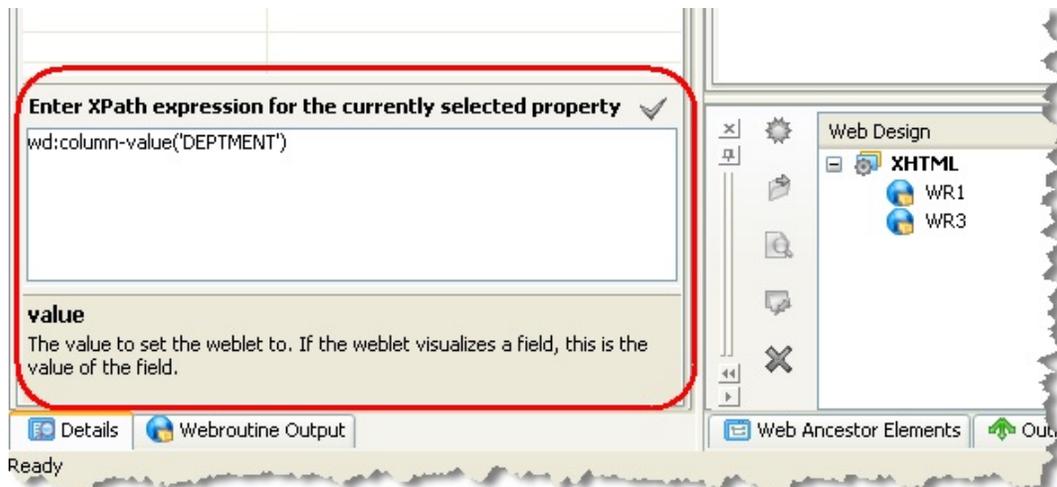
`wd:column-value('COLUMNNAME')` – returns the value of the specified column. You can type `#COLUMNNAME` into the property and the LANS editor will automatically translate it to a `wd:column-value` for you.

`wd:field-value('FIELDNAME')` – returns the value of the specified field.

`wd:variable('VARNAME')` – returns the value of the specified MTEXT or System variable.

`wd:row-index()` – returns the row number of the current row of the list. The number will be right-padded with zeros to a length of 4 digits.

These are XSL functions and should be entered in the XPath expression area.



These extensions functions need to remain in the result document after the XSL transformation is done (they are parsed by the WAM runtime after the XSL is executed). For this reason, when they are processed by the XSL processor they just echo themselves. For example, a weblet property of "{wd:column-value('COLUMNNAME')}}" results in the string "{wd:column-value('COLUMNNAME')}}" in your property variable. You can place this string directly into any HTML attribute. If you want to place it into the HTML content you should first convert it into a special tag using the *wdTagFromAttr.private* template provided in *std\_util.xsl*.

## Accessing Context Data

As with field and column values, the context information normally available in the `<lxml:context>` section of the webroutine XML cannot be accessed by XSL at runtime. Use these XSL extension functions to access context data in a weblet property:

`wd:web-user()`

`wd:webapplication()`

`wd:webapplication-title()`

`wd:webroutine()`

`wd:webroutine-title()`

`wd:service-name()`

`wd:partition()`

`wd:language()`

wd:images-path()  
wd:action-request()  
wd:layout-name()  
wd:dbcs()  
wd:align-right()  
wd:check-numeric()  
wd:debug()  
wd:trace()  
wd:task()

## Runtime data cannot be used in all properties

Some weblet properties affect how a weblet is constructed, others are passed to the browser to control CSS or JavaScript behavior. Because an inline weblet is constructed at design time, you cannot use runtime data to affect the construction. For example, see the `display_mode` property in any of the standard visualization weblets. A value of 'input' will generate an HTML `<input>` tag. A value of 'output' will generate a `<span>`. If you need to use runtime data in these properties, you should use a non-inline list.

The exceptions to this are some boolean properties like *hide\_if* and *disabled*.

## wd:boolean properties

Some boolean properties like *hide\_if* and *disabled*, whose values affect how a weblet is constructed, can still accept runtime data through the use of a string containing a *wd:boolean* expression.

The syntax of a *wd:boolean* expression is designed to optimize runtime performance, so it may not be easy to read. Properties that accept a *wd:boolean* expression will provide you with an expression editor dialog so that you don't need to be able to read the expression. Click the ellipsis button to the right of the property to open the property editor:



## Refreshing Inline Weblets

Inline Weblets used in a webroutine are not automatically updated when the weblet is changed. To update an inline weblet it must be refreshed. You can do this in several ways:

Modify a property of the weblet

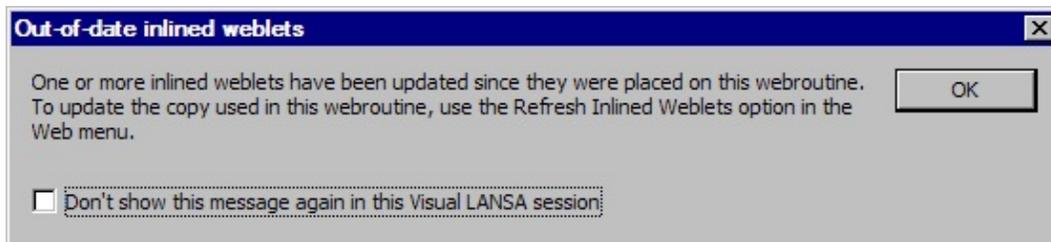
Open the Webroutine in the Design tab then right click on the weblet and select "Refresh Inlined Weblet" from the context menu.

Open the Webroutine in the Design tab and select *Refresh Inlined Weblets...* from the Web menu.

Select WAMs from the Repository or Favorites tab and select *Refresh Inlined Weblets...* from the context menu. This is the recommended way to refresh inlined weblets for a larger number of WAMs and webroutines.

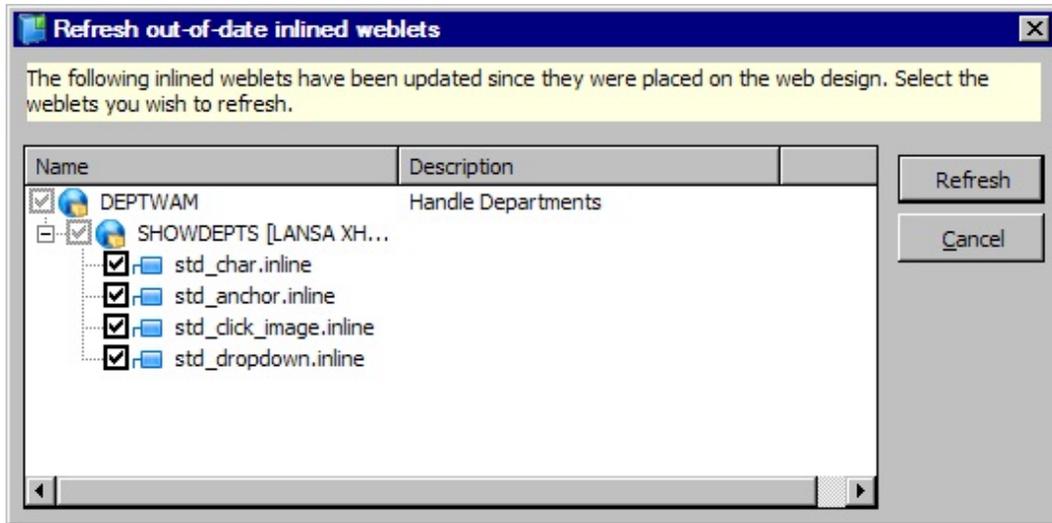
When you select WAMs from the Repository or Favorites tab every WAM and webroutines will be opened to verify if there are any inlined weblets in the webroutine and if they are out-of-date. Depending on how many WAMs and webroutines are selected, their complexity and the number of inlined weblets this process might take a while.

When you open a webroutine the LANSAs editor will tell you if it contains inlined weblets that need to be refreshed.



Note that this message only applies to the webroutine being opened. If you have multiple webroutines in a WAM you need to open each webroutine to check for differences in weblet version.

When you chose Refresh Inlined Weblets you will be able to select which weblets you wish to update.



By default all Weblets are pre-selected. It is recommended that you refresh all of them.

### **3. Essential Topics**

After you have read [An Introduction to WAMs](#) we recommend that you become familiar with the following topics:

- [3.1 Using CHECKNUMERIC in WAMs](#)
- [3.2 WAM Application Design](#)
- [3.3 Developing for Multiple Languages](#)
- [3.4 Using Cookies in Your WAM Application](#)
- [3.5 Using the Service Name](#)
- [3.7 Deleting Objects](#)
- [3.8 LOB Data Types and Stream Files](#)
- [3.9 WAM External Resources](#)
- [3.10 Using jQuery](#)
- [3.11 Theming WAMs](#)
- [3.12 Localization](#)
- [3.13 JSON Support](#)
- [3.14 Saving a WAM's Output to a File](#)
- [3.15 Document Type Declaration \(DOCTYPE\)](#)

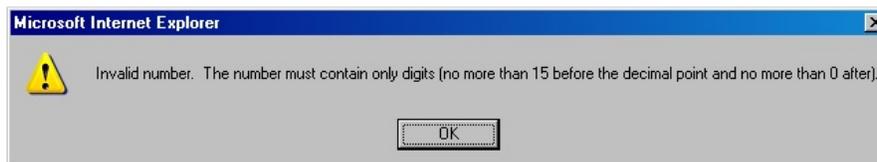
### 3.1 Using CHECKNUMERIC in WAMs

To enable numeric validation in WAMs, use the CHECKNUMERIC selector for the WAM. This is specified on the BEGIN\_COM command.

```
FUNCTION OPTIONS(*DIRECT)  
⌘ BEGIN_COM ROLE(*EXTENDS #PRIM_WAM) CheckNumeric(Yes)
```

If CHECKNUMERIC is not specified in the RDMLX code for the WAM, the default is based on the System Information value for *Web validate numerics*. To find the *Web validate numerics*, open *System Information* in the *Repository*, expand *Compile and Edit Options*. The setting is located under *Process and functions compile defaults*.

If CHECKNUMERIC is applied and an invalid number is entered at runtime the following warning is issued:



## 3.2 WAM Application Design

- When designing a LANSAs Web application, you must determine the number of WAMs required by your application and the WEBROUTINES that will belong to each WAM. Typically, you will divide your application into functional areas where each area is represented by a WAM.
- You must also understand WAM session management (refer to [WAM Session Management](#)) when designing your WAM/WEBROUTINE structures. By default, each WAM maintains its own session state independent of other WAMs, but you may chain WAM sessions.
- Modularize your WAM applications by using other RDMLX components. Use RDMLX Components in your WEBROUTINES and assign business rule processing to them. This design will provide a good foundation for code re-use across different presentation technologies, such as GUI and browser-based applications.
- Using a WEBROUTINE Service Name provides greater flexibility when deploying WAM applications. For example, it allows applications to be re-deployed to a different Partition, WAM or WEBROUTINE without having to modify any external URL references to it.

### 3.3 Developing for Multiple Languages

WAMs support the development of Web applications executing in multiple languages from a single RDMLX WAM code base. WEBROUTINE and weblet XSL stylesheets are language sensitive.

It is possible to save and deploy separate distinct stylesheet for distinct languages. For example, you can create different presentation pages and weblets for English and Chinese languages.

How you will approach the task of developing and maintaining WAMs for multiple languages depends on your application.

If your web application will have the same appearance for all languages (i.e. same menus and same page layout for the respective webrouines) the simplest way to implement a multilingual application is to supply all text values as Multilingual Variables (\*MTXT), which can be used for the appropriate captions in the weblet properties. This approach simplifies the management and distribution of your application, as only a single set of XSL source exists for each webroutine and layout.

If your web application requires a different interface for each webroutine based on the language, then you will need to create appropriate webroutine and layout XSL to support each language. This obviously is the less ideal approach as you will have more objects to maintain.

If a language specific page does not exist at execution time, a partition default page is used for presentation output. So while it is possible to have different pages for different languages, it is also possible to have a single page, which is presented for multiple languages.

Field captions and list headings are language dependent and will be presented according to the requested language. Multilingual variables are supported and can be referenced in the WEBROUTINE XSL. Where weblets allow you to specify a caption, you can specify Multilingual variable rather than specifying language dependent literal text. Weblets will display the text according to the language the WAM is being executed for.

**Important:**

1. The values of multilingual text variables referenced in webroutine and weblet XSL are resolved when the WAM or weblet is saved.
2. If you deploy a WAM or weblet, the values from the source system

are taken in the deployment package.

At runtime, a language can be requested by passing a language keyword in the URL 'lang' parameter. For example the following URL requests an English language page:

```
http://localhost/cgi-bin/lansaweb?  
srve=LEWAM01_SearchQuery&part=DEM&lang=ENG
```

For more details, refer to the *LANSA Multilingual Application Design Guide*.

### 3.4 Using Cookies in Your WAM Application

You can control how cookies are set in the browser by adding a special cookies section to your Input XML file for a WEBROUTINE. For example:

```
<lxml:server-instructions>
  <lxml:cookies>
    <lxml:cookie name="USRID">
      <lxml:value field-name="EMPNO"></lxml:value>
      <lxml:expires field-name="EXPDAT"></lxml:expires>
      <lxml:domain field-name="DOMAIN"></lxml:domain>
      <lxml:path field-name="PATH"></lxml:path>
      <lxml:secure field-name="SECFLAG"></lxml:secure>
      <lxml:httponly field-name="HTTPFLAG"></lxml:httponly>
    </lxml:cookie>
  </lxml:cookies>
</lxml:server-instructions>
```

This example demonstrates the format of the cookies section that is required if you wish to set cookies in the browser from WEBROUTINE field values. The `lxml:cookie` element contains all the required cookie information. Its name attribute specifies the name used to store the cookie, as well as the LANSAs field name that will contain the cookie value on subsequent WEBROUTINE requests.

**Note:** The cookies section must be inside the server-instructions section, as shown in the example.

The following table describes the elements:

<b>Element</b>	<b>Description</b>
<code>lxml:value</code>	The value to store for the cookie.
<code>lxml:expires</code>	The expiry date of the cookie in GMT format. If this is not specified, the cookie will expire when the browser is closed.
<code>lxml:domain</code>	The domain name to make the cookie available to all subdomains of the specified domain. If unspecified, the cookie is only available to the pages of the domain that set the cookie.
<code>lxml:path</code>	The path of the cookie even though it may be being set on a page from a different directory. This value can be used to

ensure that a cookie set in one subdirectory is available in another by specifying a path of a parent directory used by both subdirectories. If a '/' value is specified, then the cookie is available in all subdirectories of the domain. If unspecified, the cookie is only available in the path it was set in.

- lxml:secure A value of true indicates that the cookie is only available to pages from secure SSL domains.
- lxml:httponly Supported by most modern browsers. A value of true indicates that the cookie should not be accessible via non-HTTP APIs (for example, JavaScript).

The above elements may also specify a field-name attribute that contains the name of the field storing the cookie values as shown in the first example.

The following example show how cookie values can be specified directly in the XML:

```
<lxml:cookies>  
  <lxml:cookie name="LSTACT">  
    <lxml:value>Inquiry</lxml:value>  
    <lxml:expires>Thu, 31 Dec 2020 10:00:00 GMT</lxml:expires>  
    <lxml:domain>acme.com</lxml:domain>  
    <lxml:path>/home</lxml:path>  
    <lxml:secure>true</lxml:secure>  
    <lxml:httponly>true</lxml:httponly>  
  </lxml:cookie>  
</lxml:cookies>
```

## 3.5 Using the Service Name

It is possible to have different URLs (see below) to invoke the same webroutine as a webpage.

Let's look at the definition for an example webroutine, MaintainRecord:

```
* This is the entry point to the WAM.  
*  
WEBROUTINE NAME(MaintainRecord) SERVICENAME(MaintEmployee) ONENTRY(*SESSIONSTATUS_NONE)
```

Based on the WEBROUTINE definition, this webroutine can use the optional ServiceName property on the WEBROUTINE definition to invoke the WAM:

**http://<server>/cgi-bin/lansaweb?srve=MaintEmployee&part=DEX**

The ServiceName does not require further qualification, as it must be unique across the partition.

The second URL option uses the combination of WAM name and the webroutine name as a unique combination.

**http://<server>/cgi-bin/lansaweb?  
wam=WAMEX02&web rtn=MaintainRecord &part=DEX**

## 3.6 Using Session Status

Let's have a look at a simple implementation of session management.

First, set the BEGIN\_COM command in the WAM RDMLX to indicate that each webroutine in this WAM must have an active session status before it can be executed UNLESS otherwise indicated by the ONENTRY parameter associated with the specific webroutine.

```
* Session status must be active to execute the Webroutines in this WAM unless
* otherwise indicated by the OnEntry parameter of the specific WebRoutine
·BEGIN_COM ROLE(*EXTENDS #PRIM_WAM) SESSIONSTATUS(Active)
```

The entry point webroutine, in this example MaintainGrid, will be allowed to execute with no session status, as indicated by ONENTRY(\*SESSIONSTATUS\_NONE). All other webroutines in the WAM (which do not have this parameter setting) require an active session status by default.

This entry webroutine sets the session status to active and can now invoke other webroutines in the WAM, which require an active session status.

```
* This is the entry point to the WAM.
·WEBROUTINE NAME(MaintainGrid) ONENTRY(*SESSIONSTATUS_NONE)
    * set the session to active.
    #com_owner.SessionStatus := Active
    * transfer to the routine that will display the the web page.
    TRANSFER TOROUTINE(ShowGrid)
·ENDROUTINE
```

So what happens if the session status is invalid (for example, if I try to invoke another webroutine in this same WAM which does not have the ONENTRY(\*SESSIONSTATUS\_NONE) setting)?

An invalid session status fires a SessionInvalid event, which can be handled to issue the appropriate information to the user.

```
* Handle invalid session status
·EVTROUTINE HANDLING(#com_owner.SessionInvalid)
    TRANSFER TOROUTINE(#WAMEX99.SessionIsInvalid)
·ENDROUTINE
```

## 3.7 Deleting Objects

### WAMs

- When you delete a WAM from the LANSAs Editor, the WAM RDMLX and all of its WebRoutine XSL stylesheets for all languages and Technology Services are also deleted. The default WAM layout can optionally be deleted when you delete a WAM. Other weblets used by the WAM are not deleted.
- If the option to delete orphan webroutine designs when closing is enabled (the default setting), you don't need to remove them manually. Otherwise, the XSL Stylesheets are not automatically deleted, and will still appear as an item in the LANSAs Editor's Outline view and the Web Designs tab. You can delete them from the WAM Editor's Outline view.
- When you check-in a WAM, its RDMLX source and all XSL Stylesheets for all languages and Technology Services are checked-in. The WAM's layout weblet is also checked-in.
- When you check-out a WAM, its RDMLX source and all XSL Stylesheets for all languages and Technology Services are checked-out. Please note that the WAM's layout weblet is not checked-out.
- When you create a new partition language, you need to republish your WAM's webroutines. This is not done automatically.

### Weblets

- When you delete a Weblet in the WAM Editor's Weblet view, you are deleting the Weblet for all languages and all Technology Services.
- Be aware that if other Weblets or Webroutine XSL Stylesheets use a Weblet you deleted, you will get errors when opening those documents in the LANSAs Editor, or when executing the WAM that uses them. Any references to deleted Weblets must be removed by editing the Webroutine's XSL Source.
- Weblets can be checked-in or out in the LANSAs Editor's Repository view.
- When you create a new partition language, you need to republish your Weblets. This is not done automatically.

## 3.8 LOB Data Types and Stream Files

WAMs allow you to serve stream files that:

- You don't want to store on your Web server
- Documents whose contents are stored in your application data base.
- Documents created on demand.

You can serve the contents of LOB data types (BLOBs and CLOBs) and stream files located in your Application Server with special webroutines.

You create these file serving webroutines by defining the Response parameter in the Webroutine command:

```
Webroutine Name(SEND_SAMPLE) Desc('Sample Document') Response(#HTTPR)
* MYPATH is the directory where the sample documents are stored
#HTTPR.ContentFile := #MYPATH + 'sample'
Endroutine
```

The webroutine can contain RDMLX code to create the contents of the file or determine which file to send. The only requirement is that you set the ContentFile to the file name that you want to serve.

You can also send the contents of a string. In this case, you need to set the content type. The charset is assumed to be the current encoding. If that is not the case, you need to set the Charset property.

```
Webroutine Name(SEND_SAMPLE2) Desc('Sample ContentString') Response(#HTTPR2)
Define Field(#VAR) Type(*String)
#VAR := '<html><body><h1>My Page</h1></body></html>'
#HTTPR2.ContentString := #VAR
#HTTPR2.ContentType := 'text/html'
Endroutine
```

### Also see

[File Request](#)

[LOB/File Content Type](#)

[LOB/File Properties](#)

[Custom HTTP Headers](#)

[CLOBs and Files with Text Content](#)

[Compression](#)

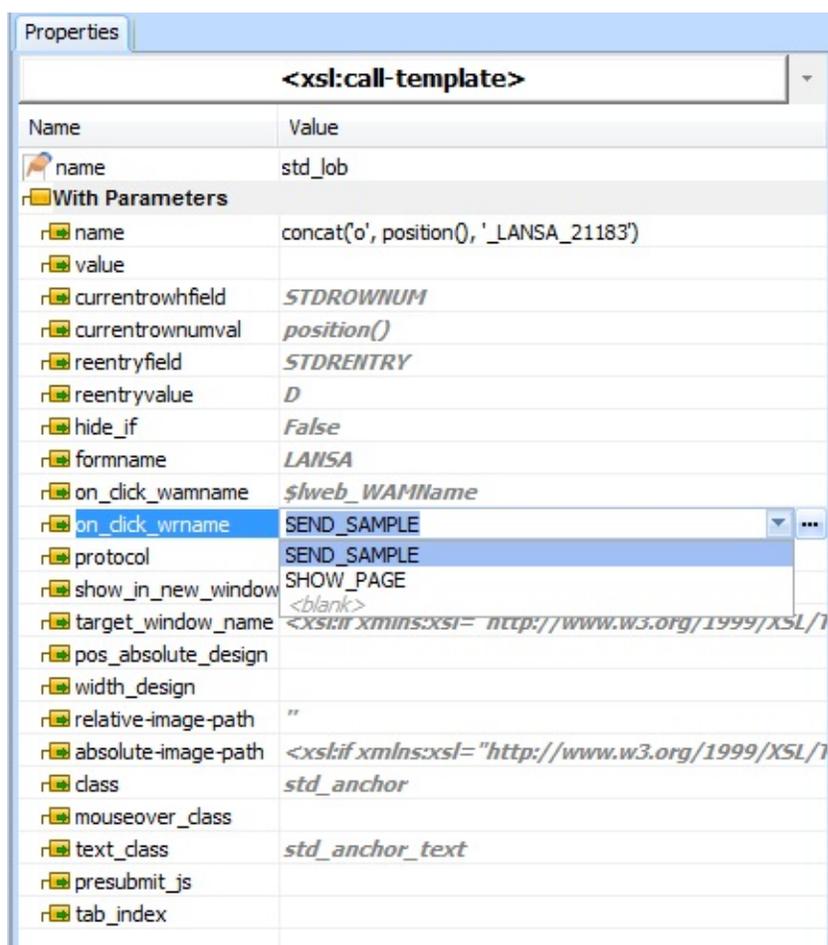
### 3.8.1 File Request

You request the LOB/file by invoking the webroutine. For example:

```
http://myhost/cgi-bin/lansaweb?  
wam=LOBSAMPLE&webrtn=SEND_SAMPLE&ml=LANSAXHTML  
&part=DEX&lang=ENG
```

You can use the Standard LOB Visualization (std\_lob) weblet if you want to show it as an anchor. You need to specify the webroutine that handles the request. You don't need to drop it on top of a LOB field. You can drop the LOB weblet on an empty space.

For example:



### **3.8.2 LOB/File Content Type**

LANSAs determines the content-type of the LOB or file based on the file extension. You normally don't need to add the content-type header. If the prefix is not known, the content type defaults to "application/octet-stream". To override the content-type, you add the content-type header as described later in this section.

### 3.8.3 LOB/File Properties

The Response variable has properties to set the following:

**ContentType:** If you need to override the default determined from the file extension or the file extension is unknown. By default LANSAs uses the file extension to determine the content type.

**Charset:** to override the character set for files with text content. By default, LANSAs determines the character set from the file's CCSID (IBM i) or the Byte Order Mark (other platforms).

**AttachmentFileName:** To ask the user to save the file as an attachment with a suggested name. If not defined, a content-disposition header is not added.

**Compression:** True/False. Use gzip encoding to compress the file. The default value is False.

**RemoveFile:** True/False. If True, the file is removed after it is sent to the user agent. The default value is False.

Note: You don't need to remove LOB files that you read from LANSAs files. They are automatically cleaned up for you by the LANSAs runtime when the request is completed.

For example:

```
* Explicit content type
#HTTPR.ContentType := 'application/pdf'

* Save file as an attachment. Suggest to save as 'mysample.pdf'
#HTTPR.AttachmentFilename := 'mysample.pdf'

* Compress file using gzip encoding
#HTTPR.Compression := True

* Remove file after it is sent
#HTTPR.RemoveFile := True
```

### 3.8.4 Custom HTTP Headers

To add other HTTP headers, use the `AddHeader` method:

```
* Custom HTTP Header example: Don't cache  
#HTTPR.AddHeader('Pragma', 'no-cache')
```

### 3.8.5 CLOBs and Files with Text Content

CLOBs and files that have text content-type are sent as text therefore the file encoding has an impact in how LANSAs handles the file.

For IBM i, LANSAs uses the file CCSID attribute to determine the character-set of the content. You can override this character-set by specifying the Charset property.

#### **Special Case for UTF-16**

The IBM i HTTP Server doesn't serve text content in UTF-16. Text content in UTF-16 is transcoded to UTF-8. The exceptions are XML documents. XML documents encoded in UTF-16 are sent with content-type application/xml (binary).

For Windows and Linux, the only encodings that are detected automatically are UTF-16 and UTF-8 files. LANSAs uses the Byte Order Mark (BOM) of the file to determine the encoding. To set the character-set for other text files, specify the Charset property of the HTTP response.

### **3.8.6 Compression**

You can compress the file by setting the Compression property or adding the HTTP header content-encoding: gzip.

Note that compression is CPU intensive and is performed each time the file is sent (that is, the compressed file is not cached). You need to balance the CPU utilization with the reduced size achieved with compression.

### **3.9 WAM External Resources**

The RDMLX Repository object *External Resource* provides the ability to store, in the Repository, all externally created resources associated with an application.

These could include

- Images
- Cascading Style Sheets (CSS)
- JavaScript files.

For Web development, LANSA uses external resources to ship:

- Themelets
- jQuery and jQuery UI Libraries
- CSS and Javascript files associated with the new jQuery Weblets.

### **Cross References**

External Resources used by WAMs and weblets appear in the cross-references. This makes it easier to include dependent objects when creating deployment packages.

### **Further Information**

[Specifying scripts and styles](#)

[Web External Resource Locations](#)

[Order of External Resource Inclusion](#)

[Shipped WAM External Resource](#)

### 3.9.1 Specifying scripts and styles

WAMs and webllets can nominate which JavaScript and CSS stylesheets they require. The JavaScript and CSS stylesheets need to be first enrolled in Visual LANSA.

Refer to the *Visual LANSA User Guide* for how to [Register Single External Resources](#) or the *Developer Guide* for how to [Register Multiple External Resources](#) and then [Using External Resources with WAMs](#) for how to link them to your Webllets.

You can add scripts and styles external resources to both webrouines and webllets. The standard practice is to add the CSS stylesheets that provide the theme (See [Theming WAMs](#)) to your site layout and scripts and other styles to your webrouines or webllets as required.

The WAM runtime consolidates the scripts and styles so that they are only added once to the output page. Refer to [Order of External Resource Inclusion](#).

[↑ 3.9 WAM External Resources](#)

## **3.9.2 Web External Resource Locations**

The location attribute determines where the web external resource goes in the web page.

### **Further information**

[Locations](#)

[Resolution of different locations](#)

## Locations

The following table lists the locations available for styles and scripts.

Type	Location	Description
Style	header	Inside the XHTML <header>
	body-top	Immediately after the XHTML <body> starts
Script	header	Inside the XHTML <header>
	body-top	Immediately after the XHTML <body> starts
	body-bottom	Just before the XHTML <body> ends
	async	Script is loaded asynchronously after the DOM is ready

Choose the location according to your needs. For example, place scripts that are not needed until the web page is complete either at the end of the page or load it asynchronously.

## **Resolution of different locations**

If a given web external resource has different locations when used by more than one weblet (or between the location provided by the weblet and that provided by the webroutine) the 'highest' precedence will be used, according to this table:

Precedence

header

body-top

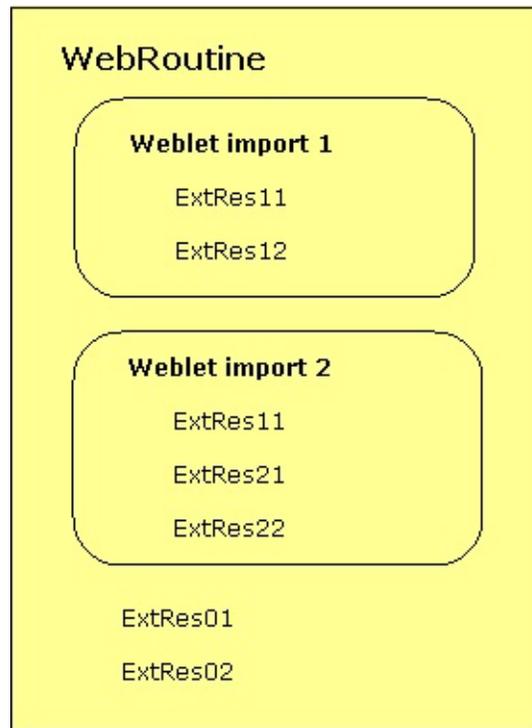
body-bottom

async

In the case of scripts, you should avoid using different locations as it might produce runtime JavaScript errors.

### 3.9.3 Order of External Resource Inclusion

The order in which scripts and styles are included is important. External resources are added in the order in which they are found. If a webroutine or weblet imports weblets that themselves have external resources, they are added at the point in which the weblet import is encountered. The following example shows the order of inclusion:



The external resources order of inclusion is:

1. ExtRes11
2. ExtRes12
3. ExtRes21 - Note that the import of External Resource 11 isn't repeated.
4. ExtRes22
5. ExtRes01
6. ExtRes02

This order ensures that you can include scripts and styles in the correct sequence when there are dependencies between the files (for example, JavaScript functions that rely on existing JavaScript libraries being present). For further details about the sequencing of external resources, refer to [Using External](#)

[Resources](#) in the *Visual LANSa User Guide*.

[↑ 3.9 WAM External Resources](#)

### 3.9.4 Shipped WAM External Resources

LANSA ships the following external resources, which are used by some of the standard weblets.

<b>Name</b>	<b>Type</b>	<b>Description</b>
XMC001	Style	Mobiscroll CSS
XMC01L	Style	jQuery Mobile Core LANSA Styles
XMCJQM	Style	jQuery Mobile Core Structural CSS
XMJ001	Script	Mobiscroll JavaScript
XMJ01L	Script	jQMobile Core LANSA Library
XMJ02L	Script	jQuery Mobile File Upload Plugin Extension
XMJJQM	Script	jQuery Mobile Core JavaScript Library
XMJQC	Script	jQuery Core JavaScript for jQMobile TSP
XMT00J	Style	jQuery Mobile Default Theme Icons
XMT01J	Style	jQuery Mobile 1.3.2 Compatible Theme
XMT02J	Style	jQuery Mobile Default Theme
XWC001	Style	jQuery Widgets Extensions CSS
XWC002	Style	jQuery TimePicker Plugin CSS
XWCFU01	Style	jQuery File Upload Plugin CSS
XWJ001	Script	jQuery Widgets Extensions
XWJ002	Script	jQuery Timepicker Plugin
XWJ003	Script	LANSA JSON JavaScript Library
XWJ004	Script	CKEditor JavaScript Library
XWJ004E	Script	CKEditor Extensions
XWJ004J	Script	CKEditor Plugin for jQuery
XWJ005	Script	Google Charts

XWJ007	Script	jQuery File Upload Plugin Extension
XWJDCJS1	Script	Douglascrockford/json-js
XWJFU01	Script	jQuery Iframe Transport Plugin
XWJFU02	Script	jQuery File Upload Plugin
XWJLLP01	Script	Lazy Loader Plugin for jQuery
XWJMODZR	Script	Modernizr JavaScript Library
XWJQC	Script	jQuery Core JavaScript Library
XWJQUI	Script	jQuery UI JavaScript Library
XWT01J	Style	Redmond – jQuery UI Widgets
XWT01L	Style	Redmond – LANSA Theme Extensions
XWT01L101	Style	Redmond – LANSA Style # 1 Themelet
XWT01L102	Style	Redmond – LANSA Style # 2 Themelet
XWT02J	Style	Pepper Grinder – jQuery UI Widgets
XWT02L	Style	Pepper Grinder – LANSA Theme Extensions
XWT02L101	Style	Pepper Grinder – LANSA Style # 1 Themelet
XWT02L102	Style	Pepper Grinder – LANSA Style # 2 Themelet
XWT03J	Style	Cupertino – jQuery UI Widgets
XWT03L	Style	Cupertino – LANSA Theme Extensions
XWT03L101	Style	Cupertino – LANSA Style # 1 Themelet
XWT03L102	Style	Cupertino – LANSA Style # 2 Themelet
XWT04J	Style	Smoothness – jQuery UI Widgets
XWT04L	Style	Smoothness – LANSA Theme Extensions
XWT04L101	Style	Smoothness – LANSA Style # 1 Themelet
XWT04L102	Style	Smoothness – LANSA Style # 2 Themelet
XWT05J	Style	UI Darkness – jQuery UI Widgets

XWT05L	Style	UI Darkness – LANSA Theme Extensions
XWT05L101	Style	UI Darkness – LANSA Style # 1 Themelet
XWT05L102	Style	UI Darkness – LANSA Style # 2 Themelet
XWT06J	Style	UI Lightness – jQuery UI Widgets
XWT06L	Style	UI Lightness – LANSA Theme Extensions
XWT06L101	Style	UI Lightness – LANSA Style # 1 Themelet
XWT06L102	Style	UI Lightness – LANSA Style # 2 Themelet
XWT07J	Style	Blitzer – jQuery UI Widgets
XWT07L	Style	Blitzer – LANSA Theme Extensions
XWT07L101	Style	Blitzer – LANSA Style # 1 Themelet
XWT07L102	Style	Blitzer – LANSA Style # 2 Themelet
XWT08J	Style	South Street – jQuery UI Widgets
XWT08L	Style	South Street – LANSA Theme Extensions
XWT08L101	Style	South Street – LANSA Style # 1 Themelet
XWT08L102	Style	South Street – LANSA Style # 2 Themelet

[↑ 3.9 WAM External Resources](#)

## 3.10 Using jQuery

### Libraries shipped with LANSA

LANSA ships the jQuery Core, jQuery UI and jQuery Mobile libraries. The JavaScript and CSS files are shipped as External Resources (See list of [Shipped WAM External Resources](#)).

### Using jQuery in your own weblets

These libraries are used by some of the standard weblets shipped with LANSA. You can also use them in your own weblets or JavaScript. You need to include the appropriate external resources either in your weblet or your layout or webroutine (according to your needs).

LANSA may upgrade the jQuery, jQuery UI and jQuery Mobile libraries in future versions of LANSA.
--

#### Also see

[jQuery Tools and Tips](#)

### 3.10.1 jQuery Tools and Tips

#### Working with other JavaScript libraries

jQuery enabled XHTML layouts have the `jQueryNoConflict` property. If set to true, `jQuery.noConflict()` is called to relinquish the `$` name. You need to include the other JavaScript library that uses the `$` name before you include jQuery Core.

#### Escaping LANS field and column names in jQuery ID selectors

LANS field names may have '\$' in their names. List column names have '.' as separators in their qualified names (for example, LIST1.0001.EMPNO). To get the field ID that you can use in a jQuery ID selector, you can use the LANS jQuery global extension `lansa.makeSafeId()`.

```
var myvar = jQuery(jQuery.lansa.makeSafeId("LIST1.0001.EMPNO"));
```

This is equivalent to:

```
var myvar = jQuery("#LIST1\\.0001\\.EMPNO");
```

### 3.11 Theming WAMs

Most weblets use the jQuery UI visual design theme, which makes it easier for you to customize colors, fonts and other visual design elements on your web pages. Themable layouts (themelets) provide a matching layout for your WAMs.

LANSA ships the following jQuery UI themes:

- Redmond
- Pepper Grinder
- Cupertino
- Smoothness
- UI Darkness
- UI Lightness
- Blitzer
- South Street

Aside from the shipped standard themelets (two different styles) you can create your own themeable layout using the [Web Application Layout Manager Wizard](#).

To implement a theme, add the jQuery UI stylesheet and either the base LANSAs Theme extensions (for example, external resource XWT01L for the Redmond theme) or the corresponding LANSAs themelet styles (for example, external resource styles XWT01J and XWT01L101 for the Redmond theme) to your WAM layout.

Alternatively, add them to your site layout so that all your WAMs share the same theme.

## 3.12 Localization

Some weblets have language dependent settings. The weblets use the ISO language or ISO language-country code for localizing text and other locale preferences, such as date and time formats. Weblets taking advantage of this setting have 'auto' default settings for localizable properties. Using these default settings is the easiest way of localizing your WAMs.

Refer to [ISO language code](#) in the *Administrator's Guide* for information about localization.

WAMs use the ISO language code nominated in the partition language definition, not the user agent's language code.

Supported languages may vary for some weblets based on third party software.

Most localizable strings are stored in JSON files in the script/i18n subdirectory of the images directory.

### Also see

[Technology Service Providers](#)

[3<sup>rd</sup> Party Libraries](#)

### 3.12.1 Technology Service Providers

Within the `18n` directory, each Technology Service has its own directory containing localizable strings.

The strings for each language are stored in a file with the name:

`std_messages-<lang-code>.json`

where `<lang-code>` is a 2 letter ISO language code followed by an optional country code. For example `"de-AT"` represents German (`"de"`) as used in Austria (`"AT"`).

All strings used by the LANSAs framework are defined in `std_messages-en.json` and this file is always loaded by the framework first. This way, if no translation is found there will at least be some string to use. Next, the framework will load and merge the language file for the current language followed by the country specific file. For example, if the partition language code is `"de-AT"` the framework will load the following files in this order:

- `std_messages-en.json`
- `std_messages-de.json`
- `std_messages-de-AT.json`

This means that when defining the `std_messages-de-AT` file, you don't need to duplicate and maintain every string. Instead, you only need to define the strings that are different from the version found in `std_messages-de.json`.

### **3.12.2 3<sup>rd</sup> Party Libraries**

3<sup>rd</sup> Party libraries may use different mechanisms to handle localizable strings. Refer to the documentation for each library for specific details. Wherever possible, 3<sup>rd</sup> party localization files will be found under scripts/i18n (e.g. jQuery) however, if specific requirements of the library prevent this, the localization files will be found within that library's home directory (e.g. jQuery UI and CKEditor).

## 3.13 JSON Support

### How to use JSON in WAMs

WAMs supports Javascript Object Notation (JSON) in two ways:

#### Posting JSON data

Refer to [JSON Convenience Wrapper](#) for details on how to post fields and lists using JSON.

#### JSON Response Webroutines

The webroutine's web maps (fields and lists) is sent as a JSON response with MIME type application/json and encoded in UTF-8. You can use these webroutines to send responses to Ajax requests.

```
Def_List Name(#list01) Fields(#deptment #deptdesc) Type(*working)

* Basic JSON response
Webroutine Name(SAMPLE1) Response(*JSON)
  Web_Map For(*output) Fields(#empno #givenname #surname #list01)

  * Prepare response
  * ...
Endroutine

* JSON response with captions
Webroutine Name(SAMPLE2) Response(*JSON) Options(*METADATA)
  Web_Map For(*output) Fields(#empno #givenname #surname #list01)

  * Prepare response
  * ...
Endroutine
```

Some of the weblets shipped by LANSA use JSON response webroutines to update data using Ajax requests.

To make it easier for you to use JSON response webroutines, convenience functions are shipped in JavaScript file `std_json.js`. To include this JavaScript file in your weblets or webroutines, add the shipped external resources `XWJJQ` (jQuery Core) and `XWJ003` (LANSA JSON Library).

This file defines JSON convenience wrapper objects that let you access a JSON response webroutine fields, lists and context information. See [JSON Convenience Wrapper](#) for details.

**Also see**

## JSON Lists

## JSON Lists

You can output a list in JSON format using the \*JSON attribute in your WEB\_MAP in a webroutine that produces a normal page.

```
Webroutine Name(SAMPLE) Desc('Sample JSON List')  
  def_list name(#list1) type(*working) fields(#deptment #deptdesc)  
  web_map for(*output) fields((#list1 *JSON))  
  * Populate list  
  * ...  
Endroutine
```

JSON lists WEB\_MAPs don't appear on the output page. You use them via JavaScript using the JavaScript object Lstd.Json.List.

### 3.13.1 JSON Convenience Wrapper

LANSAs ships a JavaScript library that provides you with an easy way to get to the fields and lists returned in JSON responses or to post data in JSON. You also use the list methods to get to JSON lists returned in a web page. To use this library, add external resources XWJDCJS1 (JavaScript file json2.js) and XWJ003 (JavaScript file std\_json.js) to your webroutine.

#### Also see

[Requesting a Webroutine](#)

[Getting Fields](#)

[Processing Lists](#)

[Getting Messages](#)

[Context Data](#)

[Building a JSON Request](#)

[Adding Fields to the JSON Request](#)

[Adding a List to the JSON Request](#)

[Defining JSON Request List Headers](#)

[Adding Entries to the JSON Request List](#)

[Posting the Json Request](#)

## Requesting a Webroutine

The `Lstd.Json.getWebroutine(options)` method provides a way for your JavaScript code to call a Webroutine. Any webrouting can be called with this method but only a JSON Response Webroutine can return data to it. The single options parameter is a JavaScript object containing zero or more of the following properties:

- Wam**            The WAM being called. If not present, the current WAM is used.
- webroutine**    The Webroutine being called. If not present, the current webroutine is used.
- Fields**        A JavaScript object containing the input values to send to the Webroutine. For example:
- ```
{
  GIVENAME: "William",
  SURNAME: "Shakespeare"
}
```
- Lists**        A JavaScript object containing lists to send to the Webroutine. Each list is an array or rows and each row is an object containing column values. For example:
- ```
{
  LIST01: [
    {DEPARTMENT: "ADM", DEPTDESC: "Administration"},
    {DEPARTMENT: "FIN", DEPTDESC: "Finance"}
  ]
}
```
- callback**     A JavaScript function that will be called when the Ajax request is completed. This function is only called on successful completion of a call to a JSON Response Webroutine. It will be passed a single parameter containing a Webroutine object that represents the Webroutine output.

Putting it all together, you get something like this:

```

/*
 * Get webroutine (Ajax request)
 * Webroutine (wr) is passed to the callback wrapped in an Lstd.Json.Wr object
 */
var options = {
  wam: "SampleWam",
  webroutine: "Sample1",
  fields: {
    GIVENAME: "John",
    SURNAME: "Smith"
  },
  lists: {
    LIST01: [
      {DEPARTMENT: "ADM", DEPTDESC: "Administration"},
      {DEPARTMENT: "FIN", DEPTDESC: "Finance"}
    ]
  },
  callback: function(wr) {
    // Code to handle the Ajax response goes here
  }
};

Lstd.Json.getWebroutine(options);

```

The Webroutine object passed to the callback function contains a number of methods for getting the fields and lists returned from the server.

## Getting Fields

Captions are only available if your webroutine has Options(\*METADATA)

```
// Get field
var empno = wr.field("EMPNO");
var empnoLabel = empno.label();
var empnoDesc = empno.description();
var empnoHeadings = empno.headings();
var empnoValue = empno.value();
```

## Processing Lists

Captions are only available if your webroutine has Options(\*METADATA)

```
// Get list
var list01 = wr.list("LIST01");

// Get list header details
var list01Hdr = list01.headers();

var deptHdr = list01Hdr.col("DEPARTMENT");
var deptHeadings = deptHdr.headings();

// Get list entries
var list01Entries = list01.entries();
var rowCount = list01Entries.rowCount();

list01Entries.each(function(entry) {
var rowNumber = entry.row();
var deptValue = entry.col("DEPARTMENT");
});
```

## Getting Messages

```
// Webroutine messages
var msgs = wr.messages();
var msgsCount = msgs.count();

// Process each message
msgs.each(function(m) {
  alert("Message: " + m);
});
```

## Context Data

WAM and webroutine context information is also available.

```
var ar = [  
  "action-request",  
  "images-path",  
  "language",  
  "partition",  
  "service-name",  
  "session-key",  
  "session-key-name",  
  "session-key-method",  
  "technology-service",  
  "user-id",  
  "webapplication",  
  "webapplication-title",  
  "webroutine",  
  "webroutine-title"];
```

```
// List context items
```

```
for (x in ar) alert(ar[x] + ": " + wr.context(ar[x]));
```

## **Building a JSON Request**

Start by creating a WebRoutine request object:

```
var wr = Lstd.Json.factory();
```

## **Adding Fields to the JSON Request**

Add/change field values. If a field is already defined, its value is replaced.

```
wr.field("EMPNO", "AA010");  
wr.field("GIVENAME", "John");  
wr.field("SURNAME", "Smith");  
wr.field("SALARY", 1000);
```

## **Adding a List to the JSON Request**

Get a JSON list object by adding a list to the WebRoutine request object.

```
var list1 = wr.addList("LIST01");
```

## **Defining JSON Request List Headers**

Define the list headers by setting the column names.

```
list1.headers(["DEPARTMENT", "DEPTDESC"]);
```

## **Adding Entries to the JSON Request List**

Add entries to the list. The column count must match the number of column names set with the headers() method.

```
var entries = list1.entries();
entries.add(["ADM", "Administration"]);
entries.add(["MKT", "Marketing"]);
```

## Posting the JSON Request

Post the request. If you don't provide a WAM name, it defaults to the current WAM.

In this example "wrb" is the WebRoutine JSON object returned by Webroutine "MyResponse".

```
wr.post({wam: "MyWam", webroutine: "MyResponse", callback:  
function(wrb) {  
// Handle the response here  
}});
```

### 3.14 Saving a WAM's Output to a File

In addition to running a WAM from a web browser, you can also run a WAM from the X\_RUN command line to save the output to a stream file.

The syntax for the X\_RUN command line for a Windows platform is:

```
X_RUN PROC=*WAMSP \  
      WMOD={wam_name} \  
      WRTN={web_routine_name} \  
      WAML={markup_language} \  
      PART={partition} \  
      LANG={language} \  
      USER={user} \  
      WASP={output_file_path}
```

Where:

the ending '\' for each line is added as a line continuation indicator to make the command line easier to read in this documentation. The above is meant to be a single command line to be submitted inside a command prompt.

**Note:** For IBM i and Linux, the actual command line required is slightly different but the X\_RUN arguments required are much the same.

The following X\_RUN arguments are essential to run a WAM and save the output to a stream file:

#### Argument Value

PROC	"*WAMSP" is the special fixed value to activate this function.
WMOD	WAM name to be executed.
WRTN	WebRoutine name in the WAM to be executed.
WAML	Markup language to run the WAM in the WMOD argument. Optional. The default is LANSA:XHTML.
PART	Partition where the WAM in the WMOD argument, belongs.
LANG	Language for running the WAM in the WMOD argument,.
USER	User for running the WAM in the WMOD argument. Optional for some platforms.
WASP	Output file path where the WAM output will be saved. The path required follows the syntax of the platform where the WAM is

executed.

For example, for Windows, you would enter:

C:\Temp\wam\_output.html (with backward slash characters).

For IBM i, it is the IFS format and with Linux, uses forward slash characters.

Additional X\_RUN arguments can be added. For example, ITRO, ITRM and ITRL can be used to enable tracing. Refer to the [X\\_RUN Parameter Summary](#) in the *Visual LANSА Technical Reference* for more information.

For example:

```
X_RUN PROC=*WAMSP WMOD=mywam WRTN=myrtn WAML=LANSА:XHTML \  
PART=DEM LANG=*DFT USER=PCXUSER WASP=C:\Temp\myrtn.html
```

Where, for Windows:

the above command line executes WebRoutine **myrtn** of WAM **mywam** in partition **DEM** using the markup language **LANSА:XHTML** and saves the output html into the stream file **C:\Temp\myrtn.html**.

For Linux, the equivalent command line would be:

```
x_run PROC=*WAMSP WMOD=mywam WRTN=myrtn WAML=LANSА:XHTML \  
PART=DEM LANG=*DFT USER=PCXUSER WASP=/tmp/myrtn.html
```

Note that the **x\_run** command is in lower case and the output file path is in the UNIX format.

For IBM i, the equivalent command line would be:

```
call x_run ('PROC=*WAMSP WMOD=mywam WRTN=myrtn WAML=LANSА:XHTML \  
PART=DEM LANG=ENG WASP=/tmp/myrtn.html')
```

There are a few limitations when running a WAM in this manner:

- As there is no interaction with a web browser, no posted data can be passed to the WAM. Similarly, any HTTP request-related information, such as HTTP cookies, are not available. What this means is that any WEB\_MAP for input fields or lists for the WebRoutine will not be updated with values, so fields would retain their default values and any lists would be empty.
- For IBM i, the output stream file will be created using the code page for the user profile used to submit the X\_RUN command.
- A WAM that creates an Active session will still create a WAM session but

that session would not be available for use by any subsequent WAM.

- A WAM that requires an Active session will get the Invalid Session event fired. For this and the reason above, WAM session management should be avoided for WAMs to be run in this manner.

### 3.15 Document Type Declaration (DOCTYPE)

This topic covers the use of the DOCTYPE declaration for directing a layout mode (what is called browser DOCTYPE "sniffing" or switching), not for DTD validation.

To add a DOCTYPE declaration to a WAM, use the `xsl:output` element. For example:

```
<xsl:output method="xml" omit-xml-declaration="yes" encoding="UTF-8"
  indent="no" doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
 />
```

To avoid conflict between the different weblets and the main webroutine XSL, we recommend you define your DOCTYPE declaration in your layout weblet only.

The LANSA shipped themelets use the strict DOCTYPE to trigger standards mode. The older layout weblets don't use any DOCTYPE.

HTML5 uses the simplified DTD-less `<!DOCTYPE html>` to trigger standards mode in browsers. XSL doesn't support this DOCTYPE declaration.

As some devices might look for the simplified DOCTYPE to detect if the document is HTML5, the WAM runtime outputs the simplified HTML5 DOCTYPE if the `xsl:output` uses the XSL-friendly `legacy-compat` DOCTYPE:

```
<xsl:output method="xml" omit-xml-declaration="yes" encoding="UTF-8"
  indent="no" doctype-system="about:legacy-compat" />
```

## **4. Advanced Topics**

4.1 TRANSFER Statements in WEBROUTINEs

4.2 CALL Statements in WEBROUTINEs

4.3 WAM Session Management

## 4.1 TRANSFER Statements in WEBROUTINEs

In its simplest form, each WEBROUTINE in a WAM is associated with one XSL stylesheet, which defines the presentation layer, also known as the Web Design. Once a WEBROUTINE is invoked, only that WEBROUTINE's page will be returned. Sometimes, however, it is necessary to "redirect" execution to a different page from RDMLX. For example, a WEBROUTINE might be requested but the session has expired. In this case, it may be necessary to display an error page or a different page to the one requested.

Another example is a Logon page with a single logon button to invoke a Logon WEBROUTINE. This WEBROUTINE authenticates the user and depending on whether the user is a registered user or a guest, it would "redirect" to the appropriate WEBROUTINE and present the appropriate page. There are many other examples where it may be necessary to "redirect" from one WEBROUTINE to another.

This "redirection" can be accomplished with the use of a TRANSFER statement. The fields are mapped to the target WEBROUTINE according to that WEBROUTINE's WEB\_MAP statements. Only fields and lists specified on the incoming WEB\_MAPs are passed to the target WEBROUTINE.

The TRANSFER statement has the following properties:

<b>Property</b>	<b>Description</b>
TOROUTINE	<p>Specify the name of a WEBROUTINE to transfer to. You can specify another WAM, in this case the WAM name followed by a WEBROUTINE name separated by a dot (e.g. #MYWAM.Browse).</p> <p>A Service Name can also be specified, if prefixed with the *SERVICE modifier.</p> <p>The value can also be provided from a field, if prefixed with the *EVALUATE modifier.</p>
OnEntry	<p>This property is used for mapping incoming fields and lists into the target WEBROUTINE. This property can be one of:</p> <ul style="list-style-type: none"><li>*MAP_NONE (does not map any fields or lists),</li><li>*MAP_ALL (maps all required fields and lists),</li><li>*MAP_LOCAL (only fields and lists on WEBROUTINE's WEB_MAPs are mapped),</li></ul>

\*MAP\_SHARED (only WAM level WEB\_MAP fields and lists are mapped, not WEBROUTINE level).

The default value is \*MAP\_ALL.

When a TRANSFER is performed, the target WEBROUTINE assumes control of execution and the source WEBROUTINE does not regain control. When the target WEBROUTINE is exited, that WEBROUTINE's page is returned, as if the WEBROUTINE was invoked directly from the browser.

The following are some examples of TRANSFER statement:

```
TRANSFER TOROUTINE(Browser)
```

```
TRANSFER TOROUTINE(#MYWAM.Browser)
```

```
TRANSFER TOROUTINE(*SERVICE EmployeeBrowse)
```

```
change #WEBRTN 'wam1.routine1'
```

```
TRANSFER TOROUTINE(*EVALUATE #WEBRTN)
```

## 4.2 CALL Statements in WEBROUTINEs

You may also use a CALL statement to redirect execution to another WEBROUTINE. The CALL statement has similar semantics to the TRANSFER statement, except the called WEBROUTINE returns control back to the calling WEBROUTINE and execution continues from that point onwards.

The CALL statement has the following properties:

<b>Property</b>	<b>Description</b>
WEBROUTINE	<p>Specify the name of a WEBROUTINE to transfer to. You can specify another WAM, in this case WAM name followed by a WEBROUTINE name separated by a dot (e.g. #MYWAM.Browse).</p> <p>A Service Name can also be specified, if prefixed with *SERVICE modifier.</p> <p>The value can also be provided from a field, if prefixed with *EVALUATE modifier.</p>
OnEntry	<p>This property is used for mapping incoming fields and list into the target WEBROUTINE. This property can be one of:</p> <ul style="list-style-type: none"><li>*MAP_NONE (does not map any fields or lists),</li><li>*MAP_ALL (maps all required fields and lists),</li><li>*MAP_LOCAL (only fields and lists on WEBROUTINE's WEB_MAPs are mapped),</li><li>*MAP_SHARED (only WAM level WEB_MAP fields and lists are mapped, not WEBROUTINE level).</li></ul> <p>The default value is *MAP_ALL.</p>
OnExit	<p>This property used for mapping outgoing fields from the target WEBROUTINE. This property can be one of:</p> <ul style="list-style-type: none"><li>*MAP_NONE (does not map any fields or lists),</li><li>*MAP_ALL (maps all required fields and lists),</li><li>*MAP_LOCAL (only fields and lists on WEBROUTINE's WEB_MAPs are mapped),</li><li>*MAP_SHARED (only WAM level WEB_MAP fields and lists are mapped, not WEBROUTINE level).</li></ul>

The default value is \*MAP\_ALL.

The following are some examples of CALL statement:

CALL WEBROUTINE(Browser)

CALL WEBROUTINE(#MYWAM.Browser)

CALL WEBROUTINE(\*SERVICE EmployeeBrowse)

CALL WEBROUTINE(\*EVALUATE #WEBRTN)

## 4.3 WAM Session Management

A user Web interaction with a server involves many requests from the same user/client to the server. Each request from a server's perspective is a unique request. The HTTP protocol is stateless and does not provide an implicit mechanism for maintaining a continuous session across multiple Web requests for the same user/client. Refer to [Being Stateless](#) for further information.

To overcome these challenges, most transactional web-based applications provide their own mechanism for session management. A WAM uses a declarative, implicit, secure session management mechanism. Session management has been designed to be as transparent as possible to the rest of the application.

Web session data or state is stored in a database on the server and is identified with a unique session key that is passed between the server and browser. Since the browser does not have a mechanism to explicitly terminate a session, a timeout mechanism is provided which expires a session once there are no more requests from the same user within a specified time period.

For more details, review the following:

[4.3.1 Session Management Configuration](#)

[4.3.2 Session Key Method](#)

[4.3.3 The WEB\\_MAP \\*PERSIST Keyword](#)

[4.3.4 Session State Maintenance](#)

[4.3.5 The Mechanics of Session Management](#)

[4.3.6 WAM Session Properties](#)

[4.3.7 WAM Session Example](#)

[4.3.8 Session States](#)

A WAM application has the facility to store session state on the server in a server independent way. To declare session state, it is necessary to set OPTIONS attribute of fields or lists to \*PERSIST. Using \*PERSIST ensures that whatever changes are made to those fields and lists, they are always accessible on the server once a WEBROUTINE is executed in the context of that session. Storing the actual session state in a database enables a multi-stream execution environment. Consequently, it is possible to setup and configure multiple application servers, on separate boxes, for greater scalability. Individual WEBROUTINE requests could be shared amongst all available servers. When a request is made, the session state (i.e. all the fields and lists

marked OPTIONS(\*PERSIST)) is reloaded from a single database into server memory before WEBROUTINE RDMLX is executed. The session state is unloaded when the WEBROUTINE exits.

### 4.3.1 Session Management Configuration

Session configuration is done by directly specifying values for WAM properties. There are properties for configuring how the session key is stored in the browser and for setting the inactivity timeout period of the session. (You will find details in [4.3.6 WAM Session Properties](#).)

By default, each WAM maintains its own session state independent of other WAMs; however, you can chain multiple WAM sessions as a single session. Chaining is useful in situations where multiple WAMs comprise a single Web application.

WAMs allow you to specify the behavior on session expiry or when a session is invalid. You can provide an event handler for the event that is signaled when a session is invalid or expired. The code in the event handler might show a custom error page, or re-direct to a logon page, or provide some other functionality.

### 4.3.2 Session Key Method

To identify a session with the client, a unique session key is allocated to the session when the session is created. This session key is returned back to the browser.

To identify that a request belongs to a particular session, the browser must pass the session key back to the server with each request. The passing of the session key is done transparently to the rest of the application.

A WAM supports three ways of maintaining the session key in the browser:

1. It can be stored in a hidden field that is returned for every page.
2. It can be passed back in a URL.
3. It can be stored in a cookie maintained in the browser memory. You have a choice of standard cookie or secure cookie. If you choose secure cookies the session key will only be passed via an SSL (HTTPS) connection to prevent any chance of eavesdropping on session key. This implies that an application using secure cookie mechanism must be served to the browser via an HTTPS protocol.

The third method, secure cookies, is considered the most secure because they are the most difficult to steal either visually or programmatically.

The hidden field Session Key Method doesn't work in jQuery Mobile because of the way pages are loaded without doing a full page refresh. Use either the URL or cookie methods instead.

### 4.3.3 The WEB\_MAP \*PERSIST Keyword

The \*PERSIST option can be used inside OPTIONS property of a WEB\_MAP. WAMs are designed to execute in a server-side Web-based environment. In order to achieve maximum scalability and minimize resource utilization, a WAM Component is stateless. Any WAM memory state is maintained only while a WEBROUTINE is being executed. At the completion of WEBROUTINE execution, any memory state (such as field and list values) are destroyed. (For more information, refer to [4.3 WAM Session Management](#).)

It is possible to declare fields and lists to retain or persist data that is to be available across WEBROUTINE executions for the duration of a Web session. Persistence is achieved by declaring an OPTIONS(\*PERSIST) on a WEB\_MAP for fields or lists. In addition to declaring an OPTIONS(\*PERSIST) the WEBROUTINE needs to have an initial SessionStatus set to Active. This is done by, either setting it for the WAM or by setting OnEntry(\*SessionStatus\_Active) keyword for individual WEBROUTINE. By setting SessionStatus to Active, in this way, ensures that the WEBROUTINE will load the session state before it starts execution.

You can use the following syntax to declare session data, which is not mapped in or out of WEBROUTINES:

```
WEB_MAP FOR(*NONE) FIELDS field and/or list names with a # prefix OP
```

When a FOR(\*NONE) declaration is used, the declared fields and lists will not be mapped in or out of WEBROUTINES, but their values will be available in WEBROUTINE RDMLX across WEBROUTINE executions. If the above WEB\_MAP is declared at WAM level, (i.e. after BEGIN\_COM but outside of any WEBROUTINE block), then every WEBROUTINE, that requires a session (i.e. its initial SessionStatus is not None), will have access to these fields and lists as they form session state.

### 4.3.4 Session State Maintenance

One of the biggest advantages of WAM session management is the ability to maintain state or data for a particular session without using server memory (which is a limited resource). WAMs provide a declarative mechanism (described in [4.3.3 The WEB\\_MAP \\*PERSIST Keyword](#)) to identify fields and lists that must be maintained for the session beyond the scope of a single request. Fields and lists declared as session data are maintained in a database and are then moved in and out of server memory for each request from a session. This mechanism allows distinct application servers (in a multiple server configuration) to service individual requests even from the same session, as long as a single database is used for session state storage.

The time between two requests from a session must be within an inactivity timeout period. Each request resets the inactivity timeout. If a request occurs outside of the timeout period for a session, the session is deemed expired and appropriate action must be taken at the application level to handle this situation. For example, an RDMLX event handler must be written to deal with an invalid session key. (If an event handler is not provided, a generic error page is returned.)

When a session has expired, it is not immediately removed from the database. It is just flagged as expired. A separate clean up process is run to perform expired session clean up. This process can be configured as a background process that can be executed from the database at off-peak times. Refer to [WAM Session Clean Up](#) in the *Web Administration Guide* for information about clearing the database.

### **4.3.5 The Mechanics of Session Management**

There is a pre-set configurable pool of Web jobs for processing WEBROUTINE requests. A Web job currently processing a request will load the necessary state and reset the internal state, and then make itself available for the next request which may come from a different Web session altogether. The load is shared by Web jobs at the individual request level and not the session level. As a result, it is possible to have a much larger number of active browser sessions than Web jobs running.

The stateless nature of WAM execution environment has an important implication. A WAM Component's state only exists for the duration of WEBROUTINE execution. Unless fields and lists are declared as \*PERSIST, their values are destroyed and not available for the next request after WEBROUTINE exit. Consequently, any other RDMLX Components instantiated during WEBROUTINE execution will be destroyed upon WEBROUTINE termination. However, any \*PERSIST fields and lists, live beyond the lifetime of a request and are still available. They can span multiple WEBROUTINE requests until session termination.

### **Session Management Summary**

Following is a summary of important features of the WAM execution environment:

- A WAM Component is a type of RDMLX Component. It can instantiate and invoke other non-visual RDMLX Components.
- Each WEBROUTINE in a WAM can be invoked from the browser. It can also be exposed via a Service Name mechanism.
- A WAM supports the concept of a Web session. The session key to identify the Web session is stored in the browser.
- When a WEBROUTINE request is made from a browser (or other type of client device), a WAM Component is created, its requested WEBROUTINE is executed, and the WAM Component is destroyed. This process reduces the server resource usage and ensures maximum scalability.
- Since WAM execution maintains state on a per-request basis, a configurable pre-defined number of Web jobs can service many Web sessions and the requests can be shared amongst multiple application server machines.
- Any data, (i.e. fields, lists, other Component state), is destroyed upon WEBROUTINE termination and is not available on subsequent request.

- Any field or list marked with `OPTIONS(*PERSIST)` to declare it as session state are not destroyed but stored in a database on the server. It is possible to change this data in one `WEBROUTINE` request and have it available for subsequent `WEBROUTINE` requests.
- If no requests have been made for a session within a specified timeout period, the session is deemed expired. Any further requests from the same session will result in a `SessionInvalid` event being triggered. An event handler for this event can be used to handle session

### 4.3.6 WAM Session Properties

Session configuration is done by directly specifying values for WAM properties. There are properties for configuring how the session key is stored in the browser, for setting the inactivity timeout period of the session and so on. By default, each WAM maintains its own session state independent of other WAMs. The following table describes session management related WAM properties:

<b>Property</b>	<b>Value</b>	<b>Description</b>
SessionKeyMethod	<i>URL, Cookie, SecureCookie, or HiddenField</i>	This is the method of storing a session identifier, known as a session key, in the browser. The secure cookie method is considered the most secure. This property is not available at runtime.
SessionStatus	<i>Active, Invalid, Expired, or None</i>	At runtime, this property can be queried to determine the status of the Web session. You can also control creation and deletion of a session by setting this property to Active or Invalid, respectively. At design time, this property can be set on a WAM wide basis, which applies to every WEBROUTINE, or for an individual WEBROUTINE by using OnEntry keyword. Setting a value of Active at design time, ensures that a session is validated before executing a WEBROUTINE. A value of None allows a WEBROUTINE to be executed, even when a session is invalid or expired. SessionStatus can then be set to Active inside this WEBROUTINE to create a new session and return the session key to the browser. This property is

		available both at design time and runtime.
SessionTimeout	A numeric value in seconds. If set to 0, a system wide default is used. If set to -1 an infinite timeout is used (i.e. there is no timeout).	<p>A timeout value determines a period of time after which a session is deemed expired if no request has been received. Any subsequent requests for WEBROUTINEs requiring a session (SessionStatus set to Active), will not execute the WEBROUTINE, but will trigger a SessionInvalid event. The timeout applies to all WEBROUTINEs in the WAM.</p> <p>The session state may still be in the session store, since you can schedule a clean up done later.</p> <p>This property is not available at runtime.</p>
SessionGroupName	Any alphanumeric value	<p>Every WAM maintains its own session state and is separate to other WAMs. If more than one WAM participates in a Web application, it is possible to have all required WAMs reference the same session state by assigning the same identifier in this property for all participating WAMs. This property is not available at runtime.</p>

All WEBROUTINEs in a WAM are assigned an initial SessionStatus value from a WAM-defined SessionStatus property. However, SessionStatus can be overridden by using an OnEntry keyword for a WEBROUTINE with the following values:

Value	Description
-------	-------------

*SessionStatus_Active	An active session is assumed for the WEBROUTINE to be entered for execution.
*SessionStatus_None	No session is required for the WEBROUTINE to be entered for execution.
*SessionStatus_Of_WAM	This is the default if the OnEntry keyword is unspecified. Assumes the value of the WAM's SessionStatus property.

### 4.3.7 WAM Session Example

It is important to understand the SessionStatus property for managing your WAM sessions.

The following is an example of a typical WAM which manages a session:

```
FUNCTION OPTIONS(*DIRECT)
BEGIN_COM ROLE(*EXTENDS #PRIM_WAM) SESSIONSTATUS(Active)
```

```
*The following line declares Session state #CUSTNAME field
WEB_MAP FOR(*NONE) FIELDS(#CUSTNAME) OPTIONS(*PERSIST)
```

```
WEBROUTINE NAME(Start) DESC('Initial Page') OnEntry(*SessionStatus_1
WEB_MAP FOR(*OUTPUT) FIELDS(#USERID #PASSWORD)
ENDROUTINE
```

```
WEBROUTINE NAME(Logon) DESC('Logon Page') OnEntry(*SessionStatus_1
WEB_MAP FOR(*INPUT) FIELDS(#USERID #PASSWORD)
```

```
...
```

```
* Some authentication logic, if authentication fails can TRANSFER back to St
```

```
* The following line will create a session, when WEBROUTINE exits
#COM_SELF.SessionStatus := Active
```

```
TRANSFER TOROUTINE(WelcomePage)
ENDROUTINE
```

```
WEBROUTINE NAME(WelcomePage) DESC('Welcome Page')
ENDROUTINE
```

```
WEBROUTINE NAME(Logoff) DESC('Logoff page')
#COM_SELF.SessionStatus := Invalid
ENDROUTINE
```

```
* The following event handler will handle invalid sessions and
```

```
* TRANSFER back to starting page, for logon
```

```
EVTROUTINE HANDLING(#COM_OWNER.SessionInvalid) OPTIONS(*N
TRANSFER TOROUTINE(Start)
ENDROUTINE
```

## END\_COM

- SessionStatus is set to Active. By default ,WEBROUTINEs in this WAM require a session to be valid before they are executed. SessionTimeout is 5 minutes.
- Two WEBROUTINEs, Start and Logon, override SessionStatus to None, by using OnEntry(\*SessionStatus\_None) keyword. The Start WEBROUTINE presents a page requesting logon details. Since this is an initial page, it must be executed whether there is a valid session or not. Hence, a SessionStatus of None.
- Logon WEBROUTINE is invoked from a Start page, when a button to logon is pressed. At this point, there is still no valid session so this WEBROUTINE must also be allowed to execute without a session.
- Logon validates the user id and password and, if valid, sets SessionStatus to Active. Setting SessionStatus from None to Active will create a new session when Logon WEBROUTINE exits.
- Logon WEBROUTINE performs a TRANSFER to WelcomPage WEBROUTINE. We now have an Active session, so WelcomePage is allowed to execute (it requires an Active session). When WelcomePage exits, a session key for the new session is passed back to the browser. This session key will be returned from now on for every new WEBROUTINE request from the same browser and allows the server to identify the session.
- There is also a SessionInvalid handler which simply TRANSFERs to Start page. This handler is invoked whenever there is an attempt to invoke a WEBROUTINE with a non-existent or invalid session key or when a valid session key is provided but the session has expired (no requests for more than 5 minutes). When a session is invalid or expired, a Start page requesting user id and password is always displayed in the browser.
- You can also provide a Logout button or menu in your WAM application so the session is explicitly invalidated. In such circumstances, the button invokes a Logoff WEBROUTINE, which sets SessionStatus to Invalid and invalidates the session on WEBROUTINE exit.

### 4.3.8 Session States

A session can be in one of four states:

<b>State</b>	<b>Meaning</b>
Invalid	The session key used to identify the session is not provided, or it does not match any session key the server knows about
Expired	The session key identifies a valid session, however the previous request from the same session has exceeded the SessionTimeout period.
Active	The session key identifies a known valid session and the last request from the same server was within the SessionTimeout period.
None	Used declaratively at design time to denote WEBROUTINEs that can execute without a valid session context. If SessionStatus is set to Active at WAM level, you must have at least one WEBROUTINE with OnEntry(*SessionStatus_None) keyword to allow entry to the WAM application to create a new session, via this WEBROUTINE.

Conceptually invalid or expired sessions are treated the same. They are both session statuses that prevent execution of WEBROUTINEs that require a valid session context (i.e. WEBROUTINEs with a SessionStatus property set to Active).

A WAM is able to handle invalid or expired sessions by providing a SessionInvalid event handler. This handler can TRANSFER to another WEBROUTINE or perform some other action to handle invalid or expired sessions. SessionInvalid handler removes the need for every WEBROUTINE to perform checks to verify session (if your application requires one). All session checking is done at WAM level and the SessionInvalid event is triggered to allow the WAM to customize invalid session handling.

If a SessionInvalid handler is not provided, the standard WAM invalid session page is returned to the browser.

When a session has expired, session state is not deleted from the session database immediately. This technique is used to reduce load on the server at peak times. The Transaction Monitor performs periodic cleanup of expired

session states. It is also possible to turn off periodic cleanup and schedule cleanup to be performed at designated off-peak times.

## **5. Execute WAM Applications**

5.1 Build or Compile your WAM

5.2 Deployment and Runtime Environment

5.3 WAM Uniform Resource Locator (URL)

5.4 Mapping Posted HTTP Data to Fields

5.5 How is the Output Presentation Created?

## 5.1 Build or Compile your WAM

You can **Build** a WAM using this icon  on your LANSAs Editor's toolbar or

you can **Compile** it using this icon .

When you use these icons, the options selected for the build will be either the defaults supplied with LANSAs or the options selected for your last compile.

### What is the difference between a Build and a Compile?

The **Build** process is optional and doesn't create any executable objects. The build process:

- saves the WAM
- performs a full function check (FFC) of your code
- If the Web Designs do not already exist:
  - generates a design (XSL Stylesheet) for the WAM Layout and
  - Web Designs (XSL Stylesheets) for the webroutines.

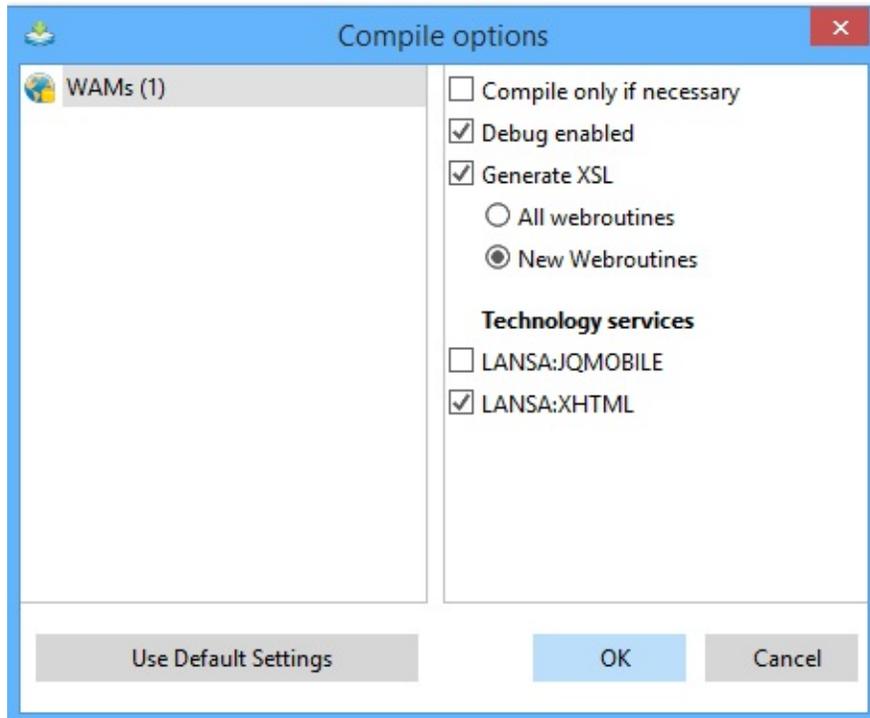
The **Compile** process is the same as a Build EXCEPT that the executable objects are produced and the webroutine XSL generation can be controlled.

You must **Compile** a WAM before it can be run, but you can do as many, or as few, **Builds** as you like.

Both the Build and Compile generate Web Designs for your **webroutines**. In addition, when required, and not found, a layout design is generated for the **WAM**. The layout design can subsequently be modified in the LANSAs Editor if required.

### Compile Options

To view or change the options for your Compile, select the Compile process from the *Verify* menu. This opens the Compile Options dialog box.



The options in this dialog are self explanatory but details can be obtained in [Component Compile Options](#) in the *LANSA Technical Reference Guide*.

The compile options selected are retained for future compiles, with the exception of the XSL Generation, which is always set to only generate New webroutines as a safeguard.

The options to be very careful about are:

### **Generate XSL**

#### *All webroutines*

If you choose this option, webroutine Web Designs will be re-generated. Any modifications that you have already made to these webroutine's designs via the LANSAs Editor will be overwritten.

#### *New webroutines*

This is the default option. This option will cause Web Designs to be generated only for new webroutines and will not overwrite any designs, which already exist.

### **Technology services**

Select the Technology Services that you will be using with this WAM.

## **5.2 Deployment and Runtime Environment**

Following is a description of the deployment and runtime environment of WAM-based Web applications.

Review the following topics:

[5.2.1 WAM and XSL Deployment](#)

[5.2.2 Multi-tier Deployment](#)

## 5.2.1 WAM and XSL Deployment

Once your WAMs are developed and the web pages have been designed and tested, the application is ready to be deployed to production. You can use the LANSAX Deployment Tool to create a deployment package.

The Deployment Tool allows you to select the WAMs to deploy and along with all of their dependent objects including the WEBROUTINE Web Designs (XSL Stylesheets).

Deploying your whole application is simply a matter of:

- selecting the necessary WAMs
- selecting all the cross-references settings to include associated objects
- selecting the Technology Service (LANSAX HTML is the default)
- creating a deployment package
- deploying the package to the production environment.

If your application uses any static external objects, such as image files, script pages or other HTML pages that are not identified to the Deployment Tool, then these objects must be deployed separately.

Refer to the WAM section of the [WAM Application with Windows Application Database](#) in the *Deployment Tool Guide*.

## 5.2.2 Multi-tier Deployment

LANSA for the Web supports both Single-tier and Multi-tier installations.

A Single-tier installation includes the Web Server and Data/Application Server on a single machine. A Multi-tier installation has the Web Server on one machine and the Data/Application Server on a separate machine.

If you install a Multi-tier configuration with a Windows IIS web server, the XSL Processing of your WAMs, by default, is moved to the Web Server tier, thereby distributing the processing load across the tiers. This option is configurable in Web Administrator, Local Configuration of your LANSAs ISAPI Plugin.

For details, refer to [Multi-Tier Web System Setup](#) in the *Web Administration Guide*.

### 5.3 WAM Uniform Resource Locator (URL)

There are two methods for invoking WEBROUTINEs from a browser using a URL.

The unique combination of the WAM and WEBROUTINE can be specified:

```
http://localhost/cgi-bin/lansaweb?wam=<WAM name>&webrtn=  
<WEBROUTINE name>&ml=<TS name>&part=  
<PARTITION name>&lang=<LANGUAGE name>
```

If a service name has been associated with a webroutine, the unique service name can be specified:

```
http://localhost/cgi-bin/lansaweb?srve=<Service name>&ml=  
<TS name>&part=<PARTITION name>&lang=<LANGUAGE name>
```

Either of the above methods to invoke a webroutine can be extended to include parameters on the URL (similar to WEBEVENTs). Any fields used as parameters on the URL must be mapped for input on the webroutine. The basic syntax is shown in the following example:

```
http://server/cgi-bin/lansaweb?  
w=wam&r=rtn&f(fieldname)=fieldvalue&f(fieldname2)=fieldvalue2
```

The following table describes each URL keyword:

<b>Keyword name</b>	<b>Short version</b>	<b>Required</b>	<b>Description</b>
field	f	Optional.	Values can be passed into a mapped field by adding the required field (fieldname)=fieldvalue parameter to the URL. For example, field(Section)=AB.
language	lang or l	Optional. If unspecified, then partition default language is used.	The language identifier determines which language is to be used for presentation of the page. It is possible to create WEBROUTINE XSL and weblet stylesheets for specific languages separate from partition default language. If a page or weblet for a specified language does not exist a partition, the default language page is returned.

markup	ml	Optional. If not specified, a configured default is used.	The Technology Service to use to render the page. In case of HTML, a default LANSAXHTML Technology Service is used. The default value can be changed using the LANSA for the Web Administrator.
partition	part or p	Optional. If not specified, a forced configured partition is always used.	The Partition identifier specifies which LANSA partition to use. If a forced Partition is configured on the server (this is done using the LANSA for the Web Administrator), this parameter is ignored.
srve	s	Mandatory. Must appear as the first keyword. Either webapp or srve keyword must be used.	The Service Name that is enrolled and mapped to a particular WAM, WEBROUTINE, Partition and Language. If this parameter is used and the specified Service Name is enrolled (can be specified in ServiceName property of a WEBROUTINE), then partition and language parameters do not need to be specified as these values are read from the registered entry. If specified, the values will override the registered entry. For further information, refer to <a href="#">Using the Service Name</a> .
webapp	wam or w	Mandatory. Must appear as the first keyword. Either webapp or srve keyword must be	The name of the WAM that contains the WEBROUTINE to be invoked.

used.

webrtn	r	Mandatory. Must appear as the second keyword. Must be used with the webapp keyword.	The name of the WEBROUTINE to be invoked.
--------	---	---	---

A similar alternative notation is also supported (for backwards compatibility with previous versions). Note the '+' used instead of the '&' to separate keyword-value pairs.

Invoking a WAM and WEBROUTINE:

```
http://localhost/cgi-bin/lansaweb?webapp=<WAM name>+webrtn=  
<WEBROUTINE name>+ml=<TS name>+part=<PARTITION name>+lang=  
<LANGUAGE name>
```

Invoking the service name:

```
http://localhost/cgi-bin/lansaweb?srve=<Service name>+ml=  
<TS name>+part=<PARTITION name>+lang=<LANGUAGE name>
```

To prevent the use of cached pages, when LANSAs issues a GET request, It appends a timestamp to the url so the request is interpreted as a request for a "different" resource. For example:

```
http://localhost/cgi-bin/lansaweb?  
wam=WAM01&webrtn=WR01&ml=LANSAXHTML&part=DEX&lang=EN  
1343366108313
```

## **5.4 Mapping Posted HTTP Data to Fields**

When a browser form is submitted to the server, the named input boxes are also submitted with the data they contain. The submitted data is in name/value pair format for each of the input boxes on the form. The server can then process the submitted values and produce the result page.

When a WEBROUTINE page is presented in the browser, that page has input boxes (or other more complex controls) for each of the fields in outgoing WEB\_MAPs. Each input box uses the same field name on the form as the field name in the WEB\_MAP. (No prefixes are used.)

When this page is submitted to another WEBROUTINE, the WAM runtime assigns the input values to the fields specified in that WEBROUTINE's incoming WEB\_MAP statements based on the field names that match the name of the input box. Hence, the posted values are assigned to appropriate fields and are available once the WEBROUTINE begins execution.

Browser Form submits Name/Value pairs

```

WEBROUTINE NAME(AddEmployee)
  WEB_MAP FOR(*BOTH) FIELDS((#SURNAME
  *OUTPUT) (#GIVENAME *OUTPUT) (#ADDRESS1
  *OUTPUT) (#ADDRESS2 *OUTPUT) (#ADDRESS3
  *OUTPUT) (#SALARY *OUTPUT))

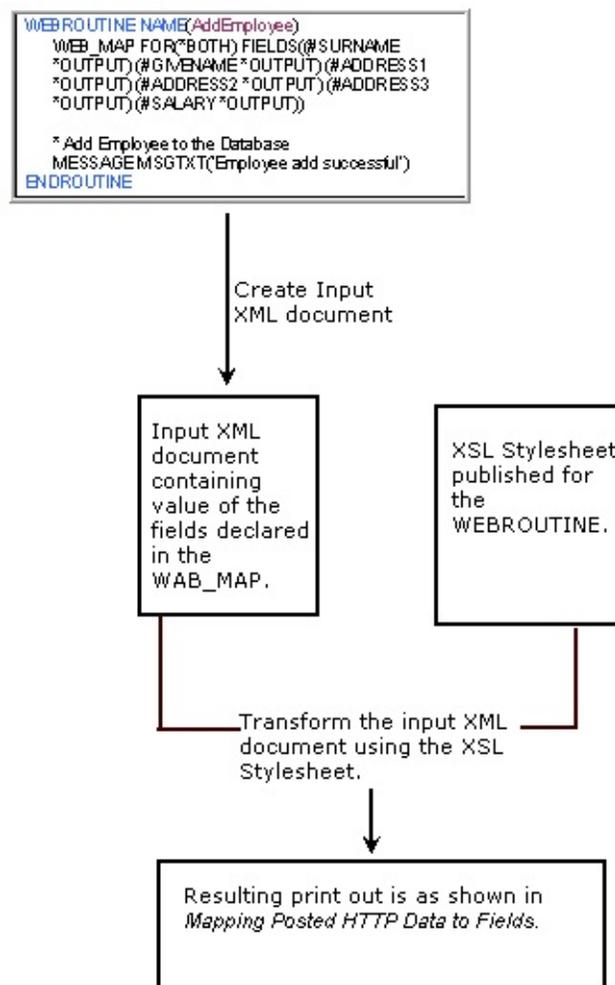
  * Add Employee to the Database
  MESSAGE MSGTXT('Employee add successful')
ENDROUTINE

```

## 5.5 How is the Output Presentation Created?

After a WEBROUTINE is invoked from the browser and the incoming fields and lists have been received (as mapped in its WEB\_MAP statements), the RDMLX code of the WEBROUTINE is executed until ENDROUTINE statement is encountered. The RDMLX logic might retrieve data from a database or perform calculations using the incoming fields.

Once the WEBROUTINE has completed, it sends back any outgoing fields and lists (as mapped in its WEB\_MAP statements) to the LANSAs Data/Application Server. The outgoing field and list values are then used to create an Input XML document in the format described in [Appendix B. WAM XML Structure](#).



The XSL Stylesheet for the WEBROUTINE, based on the required Technology Service and Language, is used to transform the Input XML document (at runtime) into the required presentation. For example, HTML is used as the

default Technology Service. The transformed presentation output (HTML) is then returned by the Web Server back to the browser.

## **6. WAM and WEBEVENT Interoperability**

LANSA V11.0 significantly extends the LANSAs ability to build web browser and internet based applications by providing a new type of component called a WAM (Web Application Module).

Prior to V11.0 the main instrument used to construct web browser applications was a special type of RDML function called a WEBEVENT function.

WEBEVENT functions continue to be fully supported and have been significantly enhanced in LANSAs V11.0. There is little or no benefit in converting existing WEBEVENT functions to WAM components. However, you may benefit by extending and enhancing your existing WEBEVENT-based web applications.

A number of techniques for constructing web applications that contain a mixture of WAM components and WEBEVENT functions are described in the following topics:

[6.1 A WAM Form Invoking a WEBEVENT Form](#)

[6.2 A WEBEVENT Form Invoking a WAM Form](#)

[6.3 A WAM Container Form Managing WEBEVENT Forms](#)

[6.4 A WEBEVENT Container Form Managing WAM Forms](#)

[6.5 Sharing Information between WAMs and WEBEVENT Functions](#)

## 6.1 A WAM Form Invoking a WEBEVENT Form

The primary mechanism for invoking a WEBEVENT form is via a JavaScript function called `HandleEvent()`. A similar JavaScript based method of invoking a WEBEVENT form from a WAM form is also provided but called `HandleWebEvent()`.

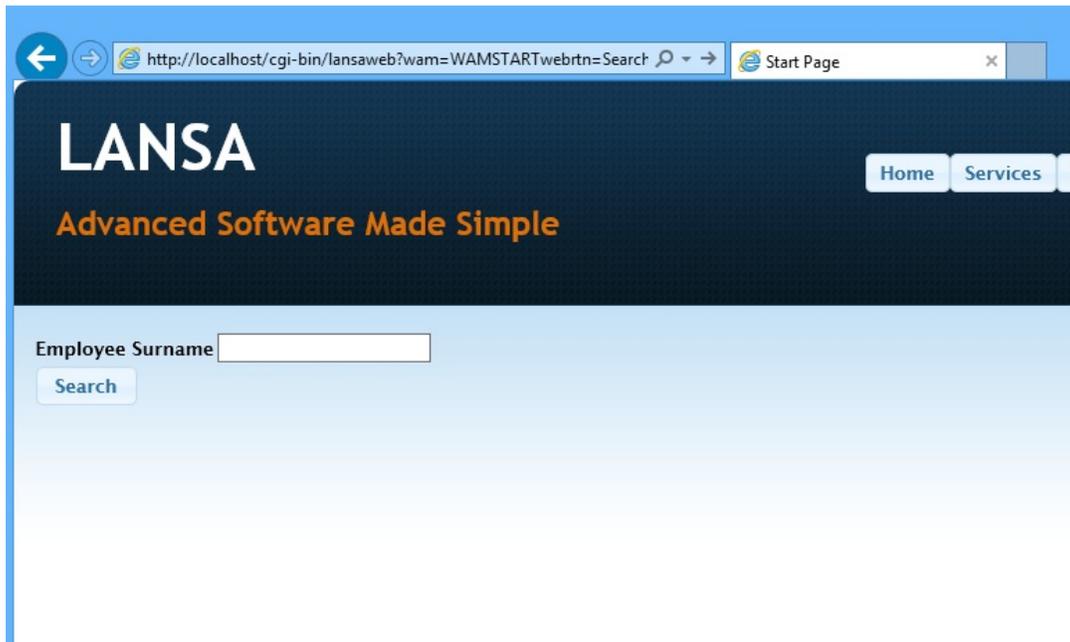
Here's how you can invoke a WEBEVENT form from a WAM form:

1. Use the JavaScript function named `HandleWebEvent()` that is provided.  
This function can be called in the same way the `HandleEvent()` function is called now.
2. There are no additional properties on weblets to navigate to a Webevent (such as there is for Webroutines). `HandleWebEvent()` can be called from the `presubmit_js` property of most weblets, or via user defined JavaScript. It is possible, for example, to directly set JavaScript to execute by setting the `onclick` attribute value, `document.LANSA.SEARCH.onclick = "HandleWebEvent('MYPROC', 'MYFUNC', null, null, 'ASURNAME', 'ASTDREENTRY')"`
3. The parameters are `HandleWebEvent(Process, Webevent, Form, Target, "ASURNAME", "ASTDREENTRY", ...)`, a variable number of parameters on the end can be passed for fields, the values of which are to be passed to the `WebEvent`. The parameters are all characters strings except for `Form`, which should be the actual form DHTML object (eg. `document.MYFORM`). It is important to provide a single character prefix before the field name. The prefix is A for Alphanumeric, P for Packed and S for Signed fields, or Q for RDMLX fields. This prefix is required so that a WEBEVENT being invoked is able to exchange the passed field values.
4. The JavaScript function gets the values for the fields from the specified `Form` parameter (or default "LANSA" form if `Form` is null), creates a temporary form and inserts the fields and their values into the temporary form for posting to the url, and then performs an HTTP post to the url. Note, the field names passed as parameters to `HandleWebEvent()` must all be prefixed with a single character prefix denoting the field type. WAM field references do not require prefixes, but WEBEVENT functions do, hence the JavaScript code retrieves the specified field values from a WAM form without this single character prefix, but posts field names to the WEBEVENT function with the prefix.

5. As a result, a WebEvent LANSa function is executed passing the specified field values and a WebEvent page is shown in the browser.

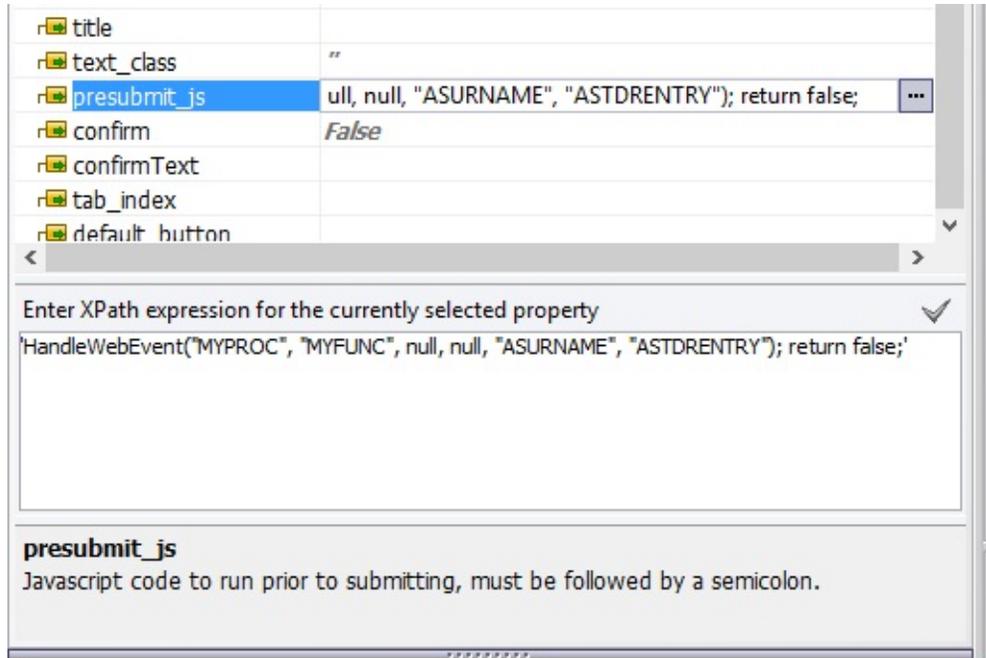
## Example

How a WAM form can initiate a WEBEVENT form and pass information to it:



Consider a Search WAM form, which submits to a Browse WEBEVENT, but passing the entered SURNAME field value.

In LANSa Editor's Details tab you enter 'HandleWebEvent("MYPROC", "MYFUNC", null, null, "ASURNAME", "ASTDRENTY"); return false;' for presubmit\_js property of the Search button.



When you run this page in the browser, clicking on Search button will submit to a WEBEVENT, showing its page in the browser, after it completes execution.

## 6.2 A WEBEVENT Form Invoking a WAM Form

The primary mechanism for invoking a WEBEVENT form is via a JavaScript function called `HandleEvent()`. A similar JavaScript-based method of invoking a WAM form from a WEBEVENT form, called `HandleWAMEvent()` is also provided.

Here's how you can invoke a WAM form from a WEBEVENT form:

1. Use the provided JavaScript function named `HandleWAMEvent()`.

This function can be called the same way `HandleEvent()` function is called now.

2. The parameters are `HandleWAMEvent(WAM, Webroutine, TechServ, Form, Target, actionRequest, Partition, Language, optSessionKey, optDebugMode, "ASURNAME", "ASTDREENTRY", ...)`, a variable number of parameters can be passed for fields, the values of which are to be passed to the Webroutine. It is important to provide a single character prefix before the field name. The prefix is A for Alphanumeric, P for Packed and S for Signed fields, or Q for RDMLX fields. Although this prefix is not required when passing field values to a WAM WEBROUTINE, it is required to access the form field value, as well as being more consistent with `HandleWebEvent()` JavaScript function semantics.

The parameters of `HandleWAMEvent` are:

WAM	The name of the target WAM.
Webroutine	The name of the target WEBROUTINE.
TechServ	The Technology Service to use, can be null for default LANSAXHTML Technology Service
Form	The form HTML object to get field values for submit from, e.g. <code>document.MYFORM</code> for a form with "MYFORM" name, can be null for default LANSAX form.
Target	The target iframe, frame or window where the result of navigation will be displayed, null to navigate to a new page.
actionRequest	If left null, is the default "cgi-bin/lansaweb" action request.
Partition	The partition to execute the WAM from.

Language            The language under which the WAM will execute.

optSessionKey    Can optionally pass the session key, if the  
SessionKeyMethod is URL, otherwise null.

optDebugMode    Can pass the debug url keyword to allow debugging of the  
WAM, otherwise null.

3. The JavaScript function gets the values for the fields from the specified Form parameter (or default "LANSA" form if Form is null), creates a temporary form and inserts the fields and their values into the temporary form for posting to the url, and then performs an HTTP post to the url. Note, the field names passed as parameters to HandleWebEvent() must all be prefixed with a single character prefix denoting the field type. WAM field references do not require prefixes, but WEBEVENT functions do, hence the JavaScript code retrieves the specified field values from a WEBEVENT form with this single character prefix, but posts field names to the WAM function without the prefix.
4. As a result a Webroutine is executed passing the specified field values and a Webroutine page is shown in the browser. The submitted fields must be specified on the WEB\_MAP FOR(\*INPUT or \*BOTH) for the values to be set in the Webroutine.

## Example

### How a WEBEVENT form can initiate a WAM form and pass information to it

1. Create a function and paste this code:

```
Function Options(*DIRECT *webevent)
*
Define Field(#searchwam) Type(*char) Length(1)
Define Field(#wamname) Type(*char) Length(9)
Define Field(#webrname) Type(*char) Length(20)
Define Field(#techserv) Type(*char) Length(21)
Define Field(#currlang) Type(*char) Length(4) Default(*language)
*
Group_By Name(#webform) Fields((#stdrentry *hidden) #surname #searchwa
*
Change Field(#wamname) To(<your wam name>)
```

Change Field(#webname) To(<your wam webroutine name>)

Change Field(#stdrentry) To(N)

\*

Request Fields(#webform) Exit\_Key(\*no) Menu\_Key(\*no) Prompt\_Key(\*no)

\*

2. Replace <your wam name> and <your webroutine name> with the appropriate names.

Note that the WAM must exist in the same partition and will execute in the same language using the default LANSA:XHTML technology service. Otherwise, change the values of fields #techserv, #curlang and #partition accordingly.

3. Using the LANSA Web Function Editor, create a Visual component type Input and call it SEARCHWAM.
4. Name the page for the component SEARCHWAM as well.
5. Create a new page and past this code:

```
<button onclick="return  
HandleWAMEvent('<RDML MERGE="WAMNAME">',  
'<RDML MERGE="WEBRNAME">', '<RDML MERGE="TECHSERV">', n
```

```
<script type="text/javascript">  
//<![CDATA[  
function CreateTempForm(ownerDoc)  
{  
    var oTempForm = ownerDoc.createElement("form");  
  
    if (oTempForm != null)  
    {  
        if (typeof oTempForm.setAttribute === "function")  
        {  
            oTempForm.setAttribute("method", "post");  
        }  
        else  
        {  
            oTempForm = ownerDoc.createElement("<form method=\"post\">  
</form>");  
        }  
    }  
}
```

```
    }  
    return oTempForm;  
}
```

```
function HandleWAMEvent(WAM, WebRoutine, techServ, Form, Target,  
actionRequest, Partition, Language, optSessionKey, optDebugMode /*, field1,  
field2, etc...*/)
{
```

```
    if (Form == null)
    {
        Form = document.LANSA;
    }
    if (techServ == null)
    {
        techServ = "LANSA:XHTML";
    }
}
```

```
var oTempForm = CreateTempForm(Form.ownerDocument);
```

```
if (oTempForm != null)
{
    Form.ownerDocument.body.appendChild(oTempForm);
    var argLen = arguments.length;

    if (argLen > 10)
    {
        for (var index = 10; index < argLen; index++)
        {
            var fieldNameWithPrefix = arguments[index];
            var fieldName = fieldNameWithPrefix.substr(1,
fieldNameWithPrefix.length - 1);
            for (var ind = fieldNameWithPrefix.length; ind < 10; ind++)
            {
                fieldNameWithPrefix += " ";
            }
            var fieldValue = Form.elements[fieldNameWithPrefix].value;
            InsertHidden(oTempForm, fieldName, fieldValue);
        }
    }
}
```

```

// Add STDANCHOR if available
var anchorField = Form.elements["ASTDANCHOR"];
if (anchorField != null)
{
    InsertHidden(oTempForm, "STDANCHOR", anchorField.value);
}

var prevAction = oTempForm.action;
var prevTarget = oTempForm.target;

var action = "";
if (actionRequest == null || actionRequest.length <= 0)
{
    actionRequest = "/cgi-bin/lansaweb";
}
action += actionRequest + "?wam=" + WAM + "&webrtn=" +
WebRoutine + "&ml=" + techServ + "&part=" + Partition + "&lang=" +
Language;
if (optDebugMode != null && optDebugMode.length > 0)
{
    action += "&debug=" + optDebugMode;
}
if (optSessionKey != null)
{
    action += "&sid=" + optSessionKey;
}
oTempForm.action = action;

if (Target != null)
{
    oTempForm.target = Target;
}
oTempForm.submit();
setTimeout(function() {
    oTempForm.action = prevAction;
    oTempForm.target = prevTarget;
    oTempForm.parentNode.removeChild(oTempForm);
}, 100);

```

```

    }
    return false;
}

function InsertHidden(Form, FieldName, FieldValue)
{
    if (Form == null)
    {
        return;
    }

    var field = Form.elements[FieldName];

    if (field == null)
    {
        var elem = Form.document.createElement("input");

        if (elem != null)
        {
            elem.setAttribute("type", "hidden");
            elem.setAttribute("name", FieldName);
            elem.setAttribute("value", FieldValue);
            Form.appendChild(elem);
        }
    }
    else
    {
        field.value = FieldValue;
    }
}
//]]>
</script>

```

6. Save the page as SEARCHWAM.
7. Compile your webevent functions generating the HTML.
8. Run this WEBEVENT example in the browser. Clicking a Search button should navigate to the WAM and WEBROUTINE you nominated in WAMNAME and WEBRNAME fields.

## 6.3 A WAM Container Form Managing WEBEVENT Forms

The primary mechanism for invoking a WEBEVENT form is via a JavaScript function called `HandleEvent()`. A similar JavaScript based method of invoking a WEBEVENT form from a WAM form, called `HandleWebEvent()` is also provided.

Here's how you can invoke a WEBEVENT form from a WAM form:

1. Use the JavaScript function named `HandleWebEvent()` that is provided.

This function can be called the same way `HandleEvent()` function is called now.

2. There are no additional properties on weblets to navigate to Webevent (such as there is for Webroutines). `HandleWebEvent()` can be called from the `presubmit_js` property of most weblets, or via user defined JavaScript. It is possible, for example, to directly set JavaScript to execute by setting the **onclick** attribute value:

```
document.LANSA.SEARCH.onclick =  
"HandleWebEvent('MYPROC', 'MYFUNC', null, 'WEFrame',  
'ASURNAME', 'ASTDRENTY')"
```

3. The parameters are `HandleWebEvent(Process, Webevent, Form, Target, "ASURNAME", "ASTDRENTY", ...)`, a variable number of parameters on the end can be passed for fields, the values of which are to be passed to the WebEvent. The parameters are all characters strings except for Form, which should be the actual form DHTML object (e.g. `document.MYFORM`). It is important to provide a single character prefix before the field name. The prefix is A for Alphanumeric, P for Packed and S for Signed fields, or Q for RDMLX fields. This prefix is required so that a WEBEVENT being invoked is able to exchange the passed field values.

The Target parameter for the JavaScript function must be specified and must be a name of the contained Navigation panel (`std_nav_panel`) weblet, window, or your own `<iframe>` or `<frame>` HTML element.

4. The JavaScript function gets the values for the fields from the specified Form parameter (or default "LANSA" form if Form is null), creates a temporary form and inserts the fields and their values into the temporary form for posting to the url, and then performs an HTTP post to the url. Note, the field names passed as parameters to `HandleWebEvent()` must all be prefixed with a

single character prefix denoting the field type. WAM field references do not require prefixes, but WEBEVENT functions do, hence the JavaScript code retrieves the specified field values from a WAM form without this single character prefix, but posts field names to the WEBEVENT function with the prefix.

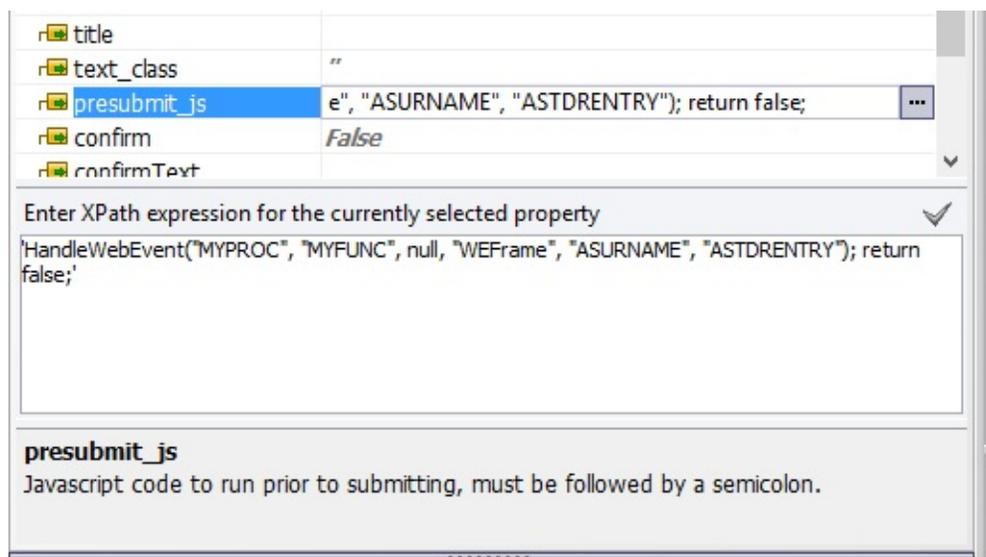
5. As a result, a WebEvent LANSA function is executed passing the specified field values and a WebEvent page is shown in the Target Navigation panel (std\_nav\_panel), window, or your own <iframe> or <frame> HTML element.

## Example

### How a WAM form can initiate a WEBEVENT form and pass information to it.

Consider a Search WAM form, which submits to a Browse WEBEVENT, but passing the entered SURNAME field value, and shows the result in Navigation panel (std\_nav\_panel) below the Search button.

1. In the LANSA Editor, drag and drop Navigation panel (std\_nav\_panel) weblet onto the page, as shown below, and enter 'WEFrame' for its name property. Select yes for its size\_panel\_to\_content property, so that it resizes itself to full size of the WEBEVENT page it navigates to.
2. Click on the Search button and enter 'HandleWebEvent("MYPROC", "MYFUNC", null, "WEFrame", "ASURNAME", "ASTDRENTY"); return false;' for the presubmit\_js property of the Search button.



3. When you run this page in the browser, clicking on Search button will submit to a WEBEVENT, showing its page in the Navigation panel (std\_nav\_panel) weblet, on the same page.

## 6.4 A WEBEVENT Container Form Managing WAM Forms

The primary mechanism for invoking a WEBEVENT form is via a JavaScript function called `HandleEvent()`. A similar JavaScript based method of invoking a WAM form from a WEBEVENT form, called `HandleWAMEvent()` is also provided.

Here's how you can invoke a WAM form from a WEBEVENT form:

1. A JavaScript function named `HandleWAMEvent()` is provided.  
This function can be called the same way `HandleEvent()` function is called now.
2. The parameters are `HandleWAMEvent(WAM, Webroutine, TechServ, Form, Target, actionRequest, Partition, Language, optSessionKey, optDebugMode, "ASURNAME", "ASTDREENTRY", ...)`, a variable number of parameters can be passed for fields, the values of which are to be passed to the Webroutine. It is important to provide a single character prefix before the field name. The prefix is A for Alphanumeric, P for Packed and S for Signed fields, or Q for RDMLX fields. Although this prefix is not required when passing field values to a WAM WEBROUTINE, it is required to access the form field value, as well as being more consistent with `HandleWebEvent()` JavaScript function semantics.
3. The parameters of `HandleWAMEvent` are:

WAM	The name of the target WAM.
Webroutine	The name of the target WEBROUTINE.
TechServ	The Technology Service to use, can be null for default LANSAXHTML Technology Service
Form	The form HTML object to get field values for submit from, e.g. <code>document.MYFORM</code> for a form with "MYFORM" name, can be null for default LANSAX form.
Target	The target iframe, frame or window where the result of navigation will be displayed, null to navigate to a new page.
actionRequest	If left null, is the default "cgi-bin/lansaweb" action request.
Partition	The partition to execute the WAM from.

Language            The language under which the WAM will execute.

optSessionKey    Can optionally pass the session key, if the SessionKeyMethod is URL, otherwise null.

optDebugMode    Can pass the debug url keyword to allow debugging of the WAM, otherwise null.

4. The JavaScript function gets the values for the fields from the specified Form parameter (or default "LANSA" form if Form is null), creates a temporary form and inserts the fields and their values into the temporary form for posting to the url, and then performs an HTTP post to the url. Note, the field names passed as parameters to HandleWebEvent() must all be prefixed with a single character prefix denoting the field type. WAM field references do not require prefixes, but WEBEVENT functions do, hence the JavaScript code retrieves the specified field values from a WEBEVENT form with this single character prefix, but posts field names to the WAM function without it.
5. The Target parameter for the JavaScript function must be specified and must be a name of the contained iframe, frame or window.
6. As a result a Webroutine is executed passing the specified field values and a Webroutine page is shown in the browser. The submitted fields must be specified on the WEB\_MAP FOR(\*INPUT or \*BOTH) for the values to be set in the Webroutine.

## Example

### How a WEBEVENT form can initiate a WAM form and pass information to it

1. Create a function and paste this code:

```
Function Options(*DIRECT *webevent)
*
Define Field(#searchwam) Type(*char) Length(1)
Define Field(#wamname) Type(*char) Length(9)
Define Field(#webname) Type(*char) Length(20)
Define Field(#techserv) Type(*char) Length(21)
Define Field(#frametgt) Type(*char) Length(20)
Define Field(#currlang) Type(*char) Length(4) Default(*language)
*
Group_By Name(#webform) Fields((#stdrentry *hidden) (#frametgt *noid) #st
```

\*

Change Field(#wamname) To(<your wam name>)

Change Field(#webname) To(<your wam webroutine name>)

Change Field(#frametgt) To(<your iframe name>)

Change Field(#stdrentry) To(N)

\*

Request Fields(#webform) Exit\_Key(\*no) Menu\_Key(\*no) Prompt\_Key(\*no)

\*

2. Replace <your wam name>, <your wam name> and <your iframe name> with the appropriate names.

Note that the WAM must exist in the same partition and will execute in the same language using the default LANSA:XHTML technology service.

Otherwise, change the values of fields #techserv, #curlang and #partition accordingly.

3. Using the LANSA Web Editor, create a Visual component type Input and call it FRAMETGT.
4. Name the page for the component FRAMETGT as well.
5. Create a new page and past this code:

```
<iframe style="width:600px; height:400px" name='<RDML MERGE="FRAM
</iframe>
```

6. Save the page as FRAMETGT.
7. Using the LANSA Web Function Editor, create a Visual component type Input and call it SEARCHWAM.
8. Name the page for the component SEARCHWAM as well.
9. Create a new page and past this code:

```
<button onclick="return
HandleWAMEvent('<RDML MERGE="WAMNAME">',
'<RDML MERGE="WEBRNAME">', '<RDML MERGE="TECHSERV">', n
```

```
<script type="text/javascript">
//<![CDATA[
```

```

function CreateTempForm(ownerDoc)
{
    var oTempForm = ownerDoc.createElement("form");

    if (oTempForm != null)
    {
        if (typeof oTempForm.setAttribute === "function")
        {
            oTempForm.setAttribute("method", "post");
        }
        else
        {
            oTempForm = ownerDoc.createElement("<form method=\"post\">
</form>");
        }
    }
    return oTempForm;
}

```

```

function HandleWAMEvent(WAM, WebRoutine, techServ, Form, Target,
actionRequest, Partition, Language, optSessionKey, optDebugMode /*, field1,
field2, etc...*/)

```

```

{
    if (Form == null)
    {
        Form = document.LANSA;
    }
    if (techServ == null)
    {
        techServ = "LANSA:XHTML";
    }
}

```

```

var oTempForm = CreateTempForm(Form.ownerDocument);

```

```

if (oTempForm != null)
{
    Form.ownerDocument.body.appendChild(oTempForm);
    var argLen = arguments.length;
}

```

```

if (argLen > 10)
{
  for (var index = 10; index < argLen; index++)
  {
    var fieldNameWithPrefix = arguments[index];
    var fieldName = fieldNameWithPrefix.substr(1,
fieldNameWithPrefix.length - 1);
    for (var ind = fieldNameWithPrefix.length; ind < 10; ind++)
    {
      fieldNameWithPrefix += " ";
    }
    var fieldValue = Form.elements[fieldNameWithPrefix].value;
    InsertHidden(oTempForm, fieldName, fieldValue);
  }
}

// Add STDANCHOR if available
var anchorField = Form.elements["ASTDANCHOR"];
if (anchorField != null)
{
  InsertHidden(oTempForm, "STDANCHOR", anchorField.value);
}

var prevAction = oTempForm.action;
var prevTarget = oTempForm.target;

var action = "";
if (actionRequest == null || actionRequest.length <= 0)
{
  actionRequest = "/cgi-bin/lansaweb";
}
action += actionRequest + "?wam=" + WAM + "&webrtn=" +
WebRoutine + "&ml=" + techServ + "&part=" + Partition + "&lang=" +
Language;
if (optDebugMode != null && optDebugMode.length > 0)
{
  action += "&debug=" + optDebugMode;
}
if (optSessionKey != null)

```

```

    {
      action += "&sid=" + optSessionKey;
    }
    oTempForm.action = action;

    if (Target != null)
    {
      oTempForm.target = Target;
    }
    oTempForm.submit();
    setTimeout(function() {
      oTempForm.action = prevAction;
      oTempForm.target = prevTarget;
      oTempForm.parentNode.removeChild(oTempForm);
    }, 100);
  }
  return false;
}

```

```

function InsertHidden(Form, FieldName, FieldValue)

```

```

{
  if (Form == null)
  {
    return;
  }

  var field = Form.elements[FieldName];

  if (field == null)
  {
    var elem = Form.document.createElement("input");

    if (elem != null)
    {
      elem.setAttribute("type", "hidden");
      elem.setAttribute("name", FieldName);
      elem.setAttribute("value", FieldValue);
      Form.appendChild(elem);
    }
  }
}

```

```
    }  
    else  
    {  
        field.value = FieldValue;  
    }  
}  
//]]>  
</script>
```

10. Save the page as SEARCHWAM.
11. Compile your webevent functions generating the HTML.
12. Run the above WEBEVENT example in the browser. Clicking a Search button should navigate to a WAM and WEBROUTINE you nominated in WAMNAME and WEBRNAME fields. The HTML response resulting from executing the WEBROUTINE will be shown in the FRAMETGT component on the same page as the WEBEVENT Search button.

## **6.5 Sharing Information between WAMs and WEBEVENT Functions**

Existing WEBEVENT functions sometimes share non-visible or server side information amongst themselves using hidden browse list(s) and/or hidden field(s) that often pass through the client browser on each web interaction. This technique is a simple form of session state management.

This technique is acceptable for sharing information between WEBEVENT functions, but it cannot be used to share information between WAMs and WEBEVENT functions - or vice-versa.

There is a newer, better and clearer general-purpose technique for sharing information between both WEBEVENT and WAM applications executing within the same web session and it is described in:

[6.5.1 Uniquely Identifying Shared Data](#)

[6.5.2 Sharing Data](#)

[6.5.3 Clean up Shared Data](#)

[6.5.4 Visual LANSA Framework and the 'Virtual Clipboard'](#)

## 6.5.1 Uniquely Identifying Shared Data

Any interaction between a browser and a server via a stateless HTTP protocol can use a unique value passed between the browser and server. This serves as special key to indentify other data maintained on the server that is never passed to the browser and not visible. It is analogous to a key in a database that identifies a row or multiple rows of data and can also server as a foreign key to other tables of data.

WAMs use a similar technique, using a unique identifier which is used as a way to determine the identity of the session and its access session state on the server.

You can use a similar method to share data between a WEBEVENT function and a WAM. All that is required is to pass a special unique identifier key between the browser and the server as the browser interaction progresses from WEBEVENT to a WAM and vice versa. The browser can simply store this value in a hidden field on the form, which is submitted with the form.

It is strongly recommended this identifier value is 32 bytes long and unique, to identify unique server data sets. WAM forms, by default, maintain and submit STDANCHOR field for this purpose. For WEBEVENT forms you should add this field to your DISPLAY/REQUEST commands to ensure it is also placed on your WEBEVENT forms. You should create this Alpha field of length 32 in your repository and use it in both your WAM and WEBEVENT code to retrieve data you wish to share between them. You can then use this field as a key into a database table that maintains shared data, or use operating system files with the key as a filename to maintain shared data.

## 6.5.2 Sharing Data

As an example, consider an application that needs to share a shopping cart. The shopping cart may be represented by a list and may contain a product identifier, product name, and quantity. Consider a WEBEVENT application where a customer adds a number of products to a shopping cart list. In your WEBEVENT code, you can assign a unique identifier to STDANCHOR and then insert entries in the shopping cart list into a database table with STDANCHOR as a key. Add STDANCHOR to the DISPLAY command to ensure that its value is available in the form as a hidden field.

When a WEBEVENT form is shown in the browser, it will also contain the STDANCHOR unique identifier. You can then navigate to a WAM using the HandleWAMEvent() JavaScript function as described in [6. WAM and WEBEVENT Interoperability](#).

HandleWAMEvent() automatically looks for the STDANCHOR field in the form and submits its value as well.

On the WAM side, the STDANCHOR field needs to be placed on the WEB\_MAP to ensure it is mapped into the WebRoutine. When the target WebRoutine is executed, it will receive the STDANCHOR identifier value and any other field values passed via HandleWAMEvent() and mapped into the WebRoutine on the WEB\_MAP. In the WebRoutine, the shopping cart list can then be retrieved from the database table using the STDANCHOR key. The values can then be placed into a WAM list that belongs to WAM Session state and the data keyed by STDANCHOR deleted from the database. Alternatively, a special cleanup job can be scheduled to run to clean up stale data from the database.

To pass data from a WAM to a WEBEVENT, the steps above can be performed in reverse order. The HandleWebEvent() JavaScript function automatically looks for the STDANCHOR field on the WAM form and submits it with the rest of submitted data.

### 6.5.3 Clean up Shared Data

As described in [6.5.2 Sharing Data](#), maintaining shared data between WAMs and WEBEVENT functions requires the data to be stored in some storage such as database or files accessible to both. The nature of browser-based applications is such that it is not always possible to know when a user has finished using the application so that appropriate clean up of shared data can be performed.

It is for this reason that all browser-based applications that require transient data to be maintained on the server need some kind of timeout mechanism to determine when this data is stale and can be cleaned up.

For this reason, when sharing data using the STDANCHOR mechanism described in [6.5.2 Sharing Data](#), it is recommended that a time stamp is also set on the data whenever shared data is stored or updated. This time stamp can then be used by the clean up job to determine the last time the data was used and whether it is a candidate for clean up.

If your application has some kind of "Logout" function, it is recommended that shared data clean up also occurs at this point. However, a user may not always perform a "Logout", hence a clean up mechanism is still required to ensure stale shared data is cleaned up.

#### **6.5.4 Visual LANSА Framework and the 'Virtual Clipboard'**

Visual LANSА Framework for the Web employs a 'virtual clipboard' facility for sharing session data. Refer to [The Virtual Clipboard](#) in the *Visual LANSА Framework Guide*.

## 7. Technology Services

WAMs use XSL Technology both for generation of XSL Stylesheets and at runtime to produce presentation output that is delivered to the browser (or other user agent).

XSL Technology allows for transformations, the output of which is another XSL Stylesheet. This is the purpose of the Technology Service XSL Stylesheet used for generation.

The Input XML to the Technology Service XSL Stylesheet is an XML file containing information about the fields and lists used in WebRoutines. This is information such as field length, field type, its attributes and so on. This information can be used by the Technology Service XSL Stylesheet to generate the appropriate WebRoutine Stylesheet to reflect its fields and lists used in WEB\_MAPs.

LANSA supplies Technology Services for XHTML and jQuery Mobile. You may need to create your own Technology Service for the following reasons:

- To implement other existing XML mark-up languages, such as BPEL, or new ones as they appear.
- To interface with a supplier or customer that expects a custom XML format.
- To customize for specific user-agents (for example, HTML with Microsoft Internet Explorer specific extensions).

To create a Technology Service, refer to

- [7.1 Create a Technology Service](#)
- [7.2 TSML Document Structure](#)
- [7.3 TSML Document Example](#)
- [7.4 WebRoutine TSP Stylesheet and the LANSA Editor](#)
- [7.5 Default Weblet for Technology Service](#)
- [7.6 About Weblets and Weblet Templates](#)

## 7.1 Create a Technology Service

To create a Technology Service, follow these steps:

[Step 1. Create a Technology Service](#)

[Step 2. Create the Technology Service XSL Stylesheets](#)

[Step 2a. Create the WebRoutine TSP Stylesheet](#)

[Step 2b. Create the Weblet TSP Stylesheet](#)

[Step 2c. Copy your Technology Service Stylesheets to the TSP directory](#)

### Step 1. Create a Technology Service

You create a Technology Service using the LANSAs Editor. The Provider and Technology Service name uniquely identify the Technology Service.

When you create a Technology Service you define its properties. The properties store definitions and options used by the LANSAs Editor and the WAM runtime.

#### Also see

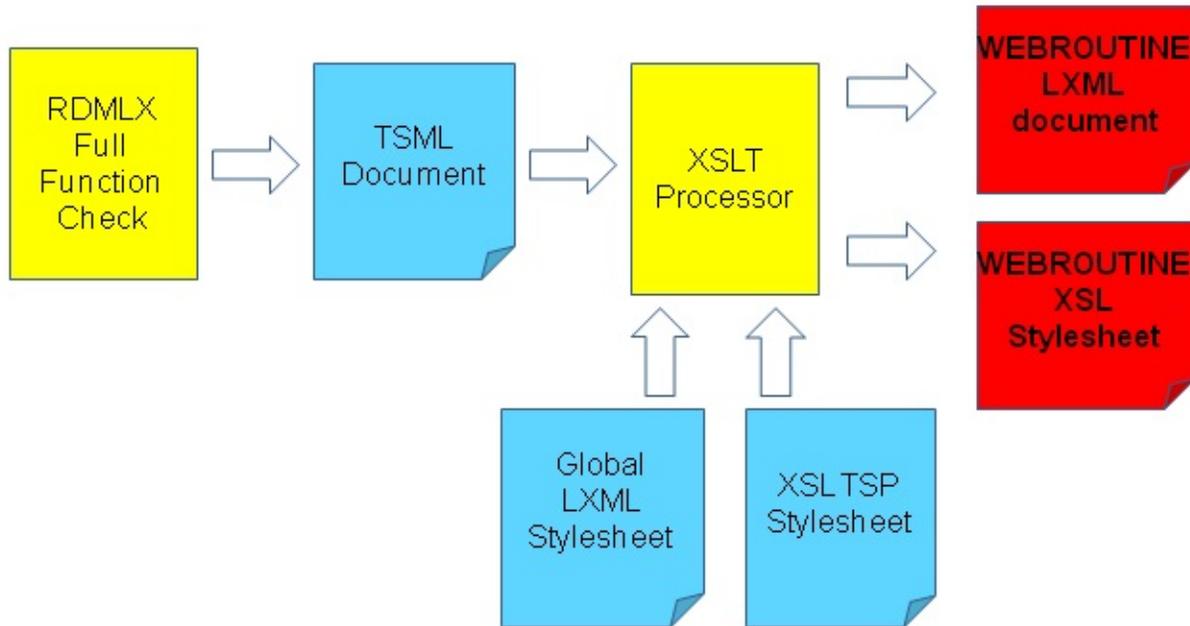
[Editing Provider Definitions](#) in the *Visual LANSAs User Guide*

[Technology Services](#) in the *Technical Reference Guide*

### Step 2. Create the Technology Service XSL Stylesheets

How WAMs use XSL stylesheets to transform the WebRoutine XML document into different presentation formats and the purpose of Technology Services, is described earlier in this guide. Refer to [WAMs Deconstructed](#).

LANSAs uses XSL stylesheets itself to generate the WebRoutine XML document and the WebRoutine XSL stylesheet for a Technology Service as shown here:



When XSL is generated for a WebRoutine, LANSAs generates an in-memory XML document named the Technology Service Markup Language (TSML) document. This is the input document used to create both the WebRoutine LXML document and the WebRoutine XSL stylesheet. A similar process is followed for the WAM layout weblet.

When you create a Technology Service you need to provide two TSP stylesheets:

- the first one to generate the WebRoutine XSL stylesheet
- the second one to generate the WAM layout weblet.

Your TSP Stylesheets must be encoded for UTF-8. That is, it should include the statement `<?xml version="1.0" encoding="UTF-8" ?>`  
 The XSL used for transformation of the XML WebRoutine document conforms with the standard W3C XSL 1.0 specification. Refer to XSL 1.0 references for information.

The Global LXML stylesheets are the same, regardless of Technology Service. You don't need to create your own.

Refer to these sections for information about the Technology Service Markup Language document:

[7.2 TSML Document Structure](#)

[7.3 TSML Document Example](#)

## Step 2a. Create the WebRoutine TSP Stylesheet

The WebRoutine TSP stylesheet is used to create the WebRoutine XSL stylesheet as shown in the diagram in Step 2.

You must follow this naming convention (Name must be all lower-case):

**tsp\_<provider>\_<technology\_service\_name>\_WebRoutine.xsl**

Where <provider> is the Technology Service Provider and <technology\_service\_name> is the Technology Service name. For example, for LANSA:XHTML the WebRoutine TSP stylesheet name is:

**tsp\_lansa\_xhtml\_WebRoutine.xsl**

The easiest way of creating a WebRoutine TSP stylesheet is to base it on the ones provided by LANSA.

The shipped WebRoutine TSP stylesheets have two top-level parameters (`g_inliner_call` and `g_import_path`). These are used to support inline lists. An inliner call is when the generator needs to insert the inline weblet.

## Step 2b. Create the Weblet TSP Stylesheet

The Weblet TSP stylesheet is used to create the WAM Layout Weblet. When you create a WAM, LANSA checks if it has a Layout weblet. If it doesn't have one, it uses this TSP stylesheet to create one.

You must follow this naming convention (Name must be all lower-case):

**tsp\_<provider>\_<technology\_service\_name>\_webletbuilder.xsl**

Where <provider> is the Technology Service Provider and <technology\_service\_name> is the Technology Service name. For example, for LANSA:XHTML the Weblet TSP stylesheet name is:

**tsp\_lansa\_xhtml\_webletbuilder.xsl**

The easiest way of creating a Weblet TSP stylesheet is to base it on the ones provided by LANSA.

**Note:** The shipped Weblet TSP stylesheets have templates for creating other weblets. Currently the only weblet you need to implement in your TSP stylesheet is the Layout weblet.

## Step 2c. Copy your Technology Service Stylesheets to the TSP directory

All TSP stylesheets must be placed in the TSP directory:

... <sysdir>\web\tsp

IBM i and Unix/Linux: ... <lansa root>/x\_lansa/web/tsp

## 7.2 TSML Document Structure

The Technology Service Markup Language (TSML) document is produced by Visual LANSA. It describes the fields and lists mapped (defined in WEB\_MAPS) in the WebRoutine, has context information from the WAM and stores values from the existing LXML document (which will be replaced) so they can be retained in the new XML/XSL to be generated. Its structure is very similar to the LXML document (which can be viewed in the XML tab of the Web Design in the LANSA Editor), but with additional meta-data used to create the XSL stylesheet.

The TSML document is divided into the following sections:

### Technology Service List

Used by Generation to determine how many XSL exist for a given Technology Service. If there is only XSL (for the default language) and the WebRoutine XSL is regenerated, then tsml nodes from the document about to be replaced can be safely removed.

### Server Instructions

These instructions are mapped directly into the LXML document. These elements are used by the WAM runtime to prepare the HTTP response.

### Weblets Section

Lists the weblets used for visualization by the **tsml:field** and **tsml:list** columns referenced in the WebRoutine. The weblets list the parameters for template calls used by each Technology Service.

### LXML Data Section

Lists content that will map into LXML data islands in XSL stylesheets. For example, it lists picklist entries for dropdowns.

### Replaced-LXML Section

This section contains the portions of the current LXML document (to be replaced by regeneration) that have values that should be retained. Currently this section includes cookies, sample messages, field captions and sample values, list captions and sample values and TSML data islands.

### Context Section

This section in the TSML document contains contextual information about the WebRoutine. Items such as WAM name, WebRoutine name, WebRoutine title are available here. **Note:** Not all the context items map into context **lxml:items**.

Some are used only during generation. For example, the **tsml:layout-name** item names the layout weblet to import in the XSL stylesheet.

### **Options Section**

The options section contains various options that may be modified for a WebRoutine that may determine whether particular validation or presentation functionality is enabled.

### **Messages Section**

Maps into the LXML document messages section.

### **Fields Section**

The fields section contains fields that appear as outgoing fields in WEB\_MAP statements in the WebRoutine. These are the fields that appear in the field list in the LXML document. In addition to the caption and value elements, the **tsml:field** has meta-data content such as display size, input-case and weblet visualization (if available).

### **Lists Section**

The lists section contains lists that appear as outgoing lists in WEB\_MAP statements in the WebRoutine. These are the lists that appear in the LXML document. In addition to the column caption and value elements, the **tsml:column** has the same meta-data content as the **tsml:field** element.

<p>The LANSAs Editor uses the WebRoutine TSP stylesheet to create XSL for fields and lists. If your Technology Service is editable by the LANSAs Editor, your templates to create fields and lists must follow the pattern used in the TSP stylesheets provided by LANSAs.</p>
--

## 7.3 TSML Document Example

The following is an example of the TSML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<tsml:data full-
document="true" inline="none" xmlns:tsml="http://www.lansa.com/2002/XMI
Metadata">
<tsml:technology-service-list>
<tsml:technology-service used_by="LANSA_XHTML" lang-count="1" />

<tsml:server-instructions>
<tsml:client-charset />
<tsml:cookies />
<tsml:ssi />
</tsml:server-instructions>

<tsml:replaced-lxml xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-
Data" />

<tsml:weblets>
<tsml:weblet name="std_dropdown.std_dropdown">
<tsml:technology-services>
<tsml:technology-service name="LANSA:XHTML" mod-
id="20120116205618000">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>display_mode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>items</tsml:param-name>
<tsml:param-role>std:picklist</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>pos_absolute</tsml:param-name>
<tsml:param-role>std:pos_absolute_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
```

```
<tsml:param-name>width_design</tsml:param-name>
<tsml:param-role>std:width_design</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
</tsml:weblets>
```

```
<tsml:lxml-data xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-
Data">
<lxml:picklist id="9C3BBEF5861148FE8B36378F5F06EF26" field-
ref="GRADEX">
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTGRADED" />
</lxml:caption>
<lxml:value>D</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTGRADEM" />
</lxml:caption>
<lxml:value>M</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTGRADEP" />
</lxml:caption>
<lxml:value>P</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTGRADEF" />
</lxml:caption>
<lxml:value>F</lxml:value>
</lxml:item>
</lxml:picklist>
</tsml:lxml-data>
```

```
<tsml:context>
<tsml:user-id>QOTHPRDOWN</tsml:user-id>
<tsml:webapplication>EMPWAM</tsml:webapplication>
<tsml:webapplication-title>Employee</tsml:webapplication-title>
<tsml:WebRoutine>skills</tsml:WebRoutine>
<tsml:WebRoutine-title>Employee skills</tsml:WebRoutine-title>
<tsml:service-name />
<tsml:partition>WEX</tsml:partition>
<tsml:language iso-lang="en">ENG</tsml:language>
<tsml:images-path>/images</tsml:images-path>
<tsml:action-request>/CGI-BIN/lansaweb</tsml:action-request>
<tsml:layout-name>empwam_layout</tsml:layout-name>
<tsml:timestamp>2012-03-07T10:30:00+10:00</tsml:timestamp>
</tsml:context>
```

```
<tsml:options>
<tsml:option name="DBCS">>false</tsml:option>
<tsml:option name="align-right">>true</tsml:option>
<tsml:option name="check-numeric">>false</tsml:option>
<tsml:option name="debug" />
<tsml:option name="trace" />
<tsml:option name="task" />
</tsml:options>
```

```
<tsml:variables />
```

```
<tsml:messages />
```

```
<tsml:fields>
<tsml:field name="EMPNO">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDE</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>5</tsml:display-max-length>
<tsml:max-length>5</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift />
```

```
</tsml:format>
<tsml:caption ref="description">
<tsml:label>Employee no....</tsml:label>
<tsml:description>Employee Number</tsml:description>
<tsml:heading-1> Employ</tsml:heading-1>
<tsml:heading-2> Number</tsml:heading-2>
<tsml:heading-3 />
</tsml:caption>
</tsml:field>
<tsml:field name="GIVENAME">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHIJKLMNQRST</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>20</tsml:display-max-length>
<tsml:max-length>20</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>O</tsml:keyboardshift>
</tsml:format>
<tsml:caption ref="description">
<tsml:label>Given names....</tsml:label>
<tsml:description>Employee Given Name(s)</tsml:description>
<tsml:heading-1>Given name(s)</tsml:heading-1>
<tsml:heading-2 />
<tsml:heading-3 />
</tsml:caption>
</tsml:field>
<tsml:field name="SURNAME">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHIJKLMNQRST</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>20</tsml:display-max-length>
<tsml:max-length>20</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>O</tsml:keyboardshift>
</tsml:format>
<tsml:caption ref="description">
<tsml:label>Surname.....</tsml:label>
```

```
<tsml:description>Employee Surname</tsml:description>
<tsml:heading-1>Surname</tsml:heading-1>
<tsml:heading-2 />
<tsml:heading-3 />
</tsml:caption>
</tsml:field>
</tsml:fields>
```

```
<tsml:lists default-sample-size="5">
<tsml:list name="SKILLS" inline="false">
<tsml:mode>input</tsml:mode>
<tsml:list-header>
<tsml:header name="SKILCODE">
<tsml:heading-1>Skill</tsml:heading-1>
<tsml:heading-2>Code</tsml:heading-2>
<tsml:heading-3 />
</tsml:header>
<tsml:header name="SKILDESC">
<tsml:heading-1>Skill</tsml:heading-1>
<tsml:heading-2>Description</tsml:heading-2>
<tsml:heading-3 />
</tsml:header>
<tsml:header name="GRADEX">
<tsml:heading-1>Grade</tsml:heading-1>
<tsml:heading-2>Obtained</tsml:heading-2>
<tsml:heading-3>for</tsml:heading-3>
</tsml:header>
<tsml:header name="DATEACQ">
<tsml:heading-1> Date Skl</tsml:heading-1>
<tsml:heading-2> Acquired</tsml:heading-2>
<tsml:heading-3 />
</tsml:header>
</tsml:list-header>
<tsml:list-entries>
<tsml:entry>
<tsml:column name="SKILCODE">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHIJ</tsml:sample-value>
<tsml:format>
```

```
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>10</tsml:display-max-length>
<tsml:max-length>10</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>O</tsml:keyboardshift>
</tsml:format>
</tsml:column>
<tsml:column name="SKILDESC">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHIJKLMNQRST</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>20</tsml:display-max-length>
<tsml:max-length>20</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>O</tsml:keyboardshift>
</tsml:format>
</tsml:column>
<tsml:column name="GRADEX">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>D</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>1</tsml:display-max-length>
<tsml:max-length>1</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift />
</tsml:format>
<tsml:use-weblets>
<tsml:use-weblet name="std_dropdown.std_dropdown" technology-
service="LANSA:XHTML" />
</tsml:use-weblets>
</tsml:column>
<tsml:column name="DATEACQ">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>12/34/56</tsml:sample-value>
<tsml:format>
<tsml:type>signed</tsml:type>
<tsml:display-max-length>11</tsml:display-max-length>
```

```
<tsml:max-length>6</tsml:max-length>
<tsml:total-digits>6</tsml:total-digits>
<tsml:fraction-digits>0</tsml:fraction-digits>
<tsml:decimal-separator>.</tsml:decimal-separator>
</tsml:format>
</tsml:column>
</tsml:entry>
</tsml:list-entries>
</tsml:list>
</tsml:lists>

</tsml:data>
```

## **7.4 WebRoutine TSP Stylesheet and the LANSA Editor**

The WebRoutine TSP stylesheet is used both during generation and by the LANSA Editor when you drag and drop fields and or lists in the web designer.

[7.4.1 Payload Wrapper XSL stylesheet](#)

[7.4.2 Sample Field Drag and Drop](#)

## 7.4.1 Payload Wrapper XSL stylesheet

The LANSAs Editor uses a Payload Wrapper" XSL stylesheet to transform a field or list TSMML node and get the contents to add to the design.

The Payload Wrapper XSL imports the WebRoutine TSP stylesheet (for the currently active TSP). The output of this XSL is the contents that the LANSAs Editor needs to add to the design (XSL imports, lxml data nodes, XSL for the field or list)

Note that the Payload Wrapper expects some XSL templates to be defined in the WebRoutine TSP Stylesheet.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- (c) 2002, 2013 LANSAs -->
<!-- WAM Editor TSP Stylesheet Wrapper -->
<!-- $Workfile:: tsp_payload_wrapper.xsl $ -->
<!-- $Revision:: 3 $ -->

<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xslt="http://www.lansa.com/2002/XSL/Transform-Alias"
  xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-Metadata"
  xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
  xmlns:lansa_design="http://www.lansa.com/2002/XML/Design"
  xmlns="http://www.w3.org/1999/xhtml"
  exclude-result-prefixes="tsml">

  <xsl:import href="%tsp_webroutine%.xsl"/>

  <xsl:output method="xml" omit-xml-declaration="no"
    encoding="UTF-8" indent="yes"/>
  <xsl:namespace-alias stylesheet-prefix="xslt" result-prefix="xsl"/>

  <xsl:template match="tsml:data[@full-document = 'false']">
  <lansa_design:payload
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
    xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
```

```
xmlns:lansa_design="http://www.lansa.com/2002/XML/Design"  
xmlns="http://www.w3.org/1999/xhtml">
```

```
<lansa_design:imports>  
<xsl:call-template name="weblet-imports"/>  
</lansa_design:imports>
```

```
<xsl:apply-templates select="tsml:lxml-data"/>
```

```
<lansa_design:content>  
<xsl:apply-templates select="*[not(self::tsml:lxml-data)]"/>  
</lansa_design:content>  
</lansa_design:payload>  
</xsl:template>
```

```
<xsl:template match="tsml:lxml-data">  
<lansa_design:lxml-data>  
<xsl:apply-imports />  
</lansa_design:lxml-data>  
</xsl:template>
```

```
<xsl:template match="tsml:fields">  
<xsl:apply-templates select="tsml:field" />  
</xsl:template>
```

```
<xsl:template match="tsml:field">  
<lansa_design:label>  
<xsl:if test="(tsml:mode != 'hidden') and (tsml:mode != 'private')">  
<xsl:call-template name="field-caption">  
<xsl:with-param name="field" select="."/>  
</xsl:call-template>  
</xsl:if>  
</lansa_design:label>  
<lansa_design:value>  
<xsl:if test="tsml:mode != 'private'">  
<xsl:call-template name="field-value">  
<xsl:with-param name="field" select="."/>  
</xsl:call-template>  
</xsl:if>
```

```

</lansa_design:value>
</xsl:template>

<xsl:template match="tsml:lists[not(@column-only)]">
<lansa_design:reference>
<xsl:apply-imports />
</lansa_design:reference>
<lansa_design:implementation>
<xsl:apply-templates select="tsml:list" mode="template_definition"/>
</lansa_design:implementation>
</xsl:template>

<xsl:template match="tsml:lists[@column-only]">
<xsl:apply-templates select="tsml:list"/>
</xsl:template>

<xsl:template match="tsml:lists[@column-only]/tsml:list">
<lansa_design:value>
<xsl:variable name="inline_list"
  select="(@inline = 'true') or ((@inline = 'default') and
$g_inline_lists)"/>
<xsl:apply-templates select="./tsml:list-entries/tsml:entry/tsml:column"
  mode="column_placement">
<xsl:with-param name="inline_list" select="$inline_list"/>
</xsl:apply-templates>
</lansa_design:value>
</xsl:template>
</xsl:transform>

```

## 7.4.2 Sample Field Drag and Drop

The following samples show the field, list and list column TSMML nodes and the resulting documents produced by by the Payload Wrapper XSL stylesheet. We use Technology Service XHTML for these samples:

[Dragging and Dropping a Field](#)

[Dragging and Dropping a List](#)

[Dragging and Dropping a List Column](#)

## **Dragging and Dropping a Field**

[Field TSML Node](#)

[Field Drag and Drop output](#)

## Field TSML Node

This is the TSML Node for a field with a picklist visualized with weblet std\_dropdown:

```
<?xml version="1.0" encoding="UTF-8"?>

<tsml:data full-document="false" inline="none"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-
  Metadata">

  <tsml:weblets>
  <tsml:weblet name="std_dropdown.std_dropdown">
  <tsml:technology-services>
  <tsml:technology-service name="LANSA:XHTML"
    mod-id="20121221132340000">
  <tsml:template-params>
  <tsml:template-param>
  <tsml:param-name>display_mode</tsml:param-name>
  <tsml:param-role>std:display_mode</tsml:param-role>
  <tsml:param-select>'input'</tsml:param-select>
  </tsml:template-param>
  <tsml:template-param>
  <tsml:param-name>items</tsml:param-name>
  <tsml:param-role>std:picklist</tsml:param-role>
  <tsml:param-select>document(")/* /lxml:data/lxml:dropdown</tsml:param-
  select>
  </tsml:template-param>
  <tsml:template-param>
  <tsml:param-name>name</tsml:param-name>
  </tsml:template-param>
  <tsml:template-param>
  <tsml:param-name>pos_absolute</tsml:param-name>
  <tsml:param-role>std:pos_absolute_design</tsml:param-role>
  </tsml:template-param>
  <tsml:template-param>
  <tsml:param-name>width_design</tsml:param-name>
  <tsml:param-role>std:width_design</tsml:param-role>
```

```
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
</tsml:weblets>
```

```
<tsml:lxml-data xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data">
```

```
<lxml:picklist id="380F247733D94ECDA37898AA9AEFCCD5"
  field-ref="DAYOFWEEK">
```

```
<lxml:item default="true">
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN05801" /></lxml:caption>
```

```
<lxml:value>MON</lxml:value>
```

```
</lxml:item>
```

```
<lxml:item>
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN06001" /></lxml:caption>
```

```
<lxml:value>TUE</lxml:value>
```

```
</lxml:item>
```

```
<lxml:item>
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN06201" /></lxml:caption>
```

```
<lxml:value>WED</lxml:value>
```

```
</lxml:item>
```

```
<lxml:item>
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN06401" /></lxml:caption>
```

```
<lxml:value>THU</lxml:value>
```

```
</lxml:item>
```

```
<lxml:item>
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN06601" /></lxml:caption>
```

```
<lxml:value>FRI</lxml:value>
```

```
</lxml:item>
```

```
<lxml:item>
```

```
<lxml:caption>
```

```
<lxml:variable name="MTXTDEMCALEN06801" /></lxml:caption>
```

```
<lxml:value>SAT</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTEXTDEMCALEN07001" /></lxml:caption>
<lxml:value>SUN</lxml:value>
</lxml:item>
</lxml:picklist>
</tsml:lxml-data>
```

```
<tsml:fields>
<tsml:field name="DAYOFWEEK">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>MON</tsml:sample-value>
<tsml:format>
<tsml:type>alpha</tsml:type>
<tsml:display-max-length>3</tsml:display-max-length>
<tsml:max-length>3</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift />
</tsml:format>
<tsml:caption ref="description">
<tsml:label>Day of the week</tsml:label>
<tsml:description>Day of the week</tsml:description>
<tsml:heading-1>Day</tsml:heading-1>
<tsml:heading-2>of</tsml:heading-2>
<tsml:heading-3>the</tsml:heading-3>
</tsml:caption>
<tsml:use-weblets>
<tsml:use-weblet name="std_dropdown.std_dropdown"
    technology-service="LANSA:JQMOBILE" />
<tsml:use-weblet name="std_dropdown.std_dropdown"
    technology-service="LANSA:XHTML" />
</tsml:use-weblets>
</tsml:field>
</tsml:fields>
</tsml:data>
```

## Field Drag and Drop output

The output of the transformation is shown . The LANSa Editor puts the relevant sections in their appropriate location in the target document (the WebRoutine XSL stylesheet)

```
<?xml version="1.0" encoding="UTF-8"?>
<lansa_design:payload xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
  xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
  xmlns:lansa_design="http://www.lansa.com/2002/XML/Design"
  xmlns="http://www.w3.org/1999/xhtml">
<lansa_design:imports>
<xsl:import href="std_dropdown.xsl" />
</lansa_design:imports>
<lansa_design:lxml-data>
<lxml:data>
<lxml:picklist id="380F247733D94ECDA37898AA9AEFCCD5">
<lxml:item default="true">
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN05801"></lxml:variable>
</lxml:caption>
<lxml:value>MON</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN06001"></lxml:variable>
</lxml:caption>
<lxml:value>TUE</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN06201"></lxml:variable>
</lxml:caption>
<lxml:value>WED</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN06401"></lxml:variable>
```

```

</lxml:caption>
<lxml:value>THU</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN06601"></lxml:variable>
</lxml:caption>
<lxml:value>FRI</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN06801"></lxml:variable>
</lxml:caption>
<lxml:value>SAT</lxml:value>
</lxml:item>
<lxml:item>
<lxml:caption>
<lxml:variable name="MTXTDEMCALEN07001"></lxml:variable>
</lxml:caption>
<lxml:value>SUN</lxml:value>
</lxml:item>
</lxml:picklist>

</lxml:data>

</lansa_design:lxml-data>
<lansa_design:content>
<lansa_design:label>
<label class="caption" for="DAYOFWEEK">
<xsl:value-of select="key('field-caption', 'DAYOFWEEK')/lxml:description"
/>
</label>
</lansa_design:label>
<lansa_design:value>
<xsl:call-template name="std_dropdown">
<xsl:with-param name="name" select="'DAYOFWEEK'" />
<xsl:with-param name="value" select="key('field-value', 'DAYOFWEEK')" />
<xsl:with-param name="display_mode" select="'input'" />
<xsl:with-param name="items"

```

```
select="document('')/*/*/lxml:data/lxml:picklist[@id =  
'380F247733D94ECDA37898AA9AEFCCD5']" />  
</xsl:call-template>  
</lansa_design:value>  
</lansa_design:content>  
</lansa_design:payload>
```

## **Dragging and Dropping a List**

List TSML Node

List Drag and Drop Output

## List TSML Node

```
<?xml version="1.0" encoding="UTF-8"?>
<tsml:data full-document="false"
  inline="none"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-
Metadata">
```

```
<tsml:weblots>
<tsml:weblet name="std_boolean.std_boolean">
<tsml:technology-services>
<tsml:technology-service name="LANSA:XHTML"
  mod-id="20121220163646000"
  proxy-format="__,_PROXY">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>display_mode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>pos_absolute</tsml:param-name>
<tsml:param-role>std:pos_absolute_design</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
<tsml:weblet name="std_input.std_input">
<tsml:technology-services>
<tsml:technology-service name="LANSA:JQMOBILE"
  mod-id="20130301081954000">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>displayMode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>id</tsml:param-name>
```

```
<tsml:param-select>concat($lweb_WRName,'_',$name)</tsml:param-select>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>label</tsml:param-name>
<tsml:param-role>std:field_caption</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>maxlength</tsml:param-name>
<tsml:param-role>std:field_maxlength</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>name</tsml:param-name>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>rdmlxDataType</tsml:param-name>
<tsml:param-role>std:rdmlx_data_type</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>size</tsml:param-name>
<tsml:param-role>std:field_display_length</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>type</tsml:param-name>
<tsml:param-role>std:field_input_type</tsml:param-role>
<tsml:param-select>'text'</tsml:param-select>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
<tsml:weblet name="std_datepicker.std_datepicker">
<tsml:technology-services>
<tsml:technology-service name="LANSA:XHTML"
  mod-id="20121220142947000"
  proxy-format="__,_PROXY">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>allow_sqlnull</tsml:param-name>
<tsml:param-role>std:allow_sqlnull</tsml:param-role>
```

```
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>display_mode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>pos_absolute</tsml:param-name>
<tsml:param-role>std:pos_absolute_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>size</tsml:param-name>
<tsml:param-role>std:field_size</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>width</tsml:param-name>
<tsml:param-role>std:width_design</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
<tsml:weblet name="std_char.std_char">
<tsml:technology-services>
<tsml:technology-service name="LANSA:XHTML"
    mod-id="20121220115335000">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>class</tsml:param-name>
<tsml:param-role>std:field_css_class</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>display_length</tsml:param-name>
<tsml:param-role>std:field_display_length</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>display_mode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
```

```
<tsml:param-name>height</tsml:param-name>
<tsml:param-role>std:height_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>keyboard_shift</tsml:param-name>
<tsml:param-role>std:keyboard_shift</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>maxlength</tsml:param-name>
<tsml:param-role>std:field_maxlength</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>pos_absolute</tsml:param-name>
<tsml:param-role>std:pos_absolute_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>type</tsml:param-name>
<tsml:param-role>std:field_input_type</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>width</tsml:param-name>
<tsml:param-role>std:width_design</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
</tsml:weblets>
```

```
<tsml:lists default-sample-size="5">
<tsml:list name="LIST01" inline="false">
<tsml:mode>input</tsml:mode>
<tsml:list-header>
<tsml:header name="BOOL01">
<tsml:heading-1>Boolean</tsml:heading-1>
<tsml:heading-2>field</tsml:heading-2>
<tsml:heading-3 /></tsml:header>
<tsml:header name="DAT01">
<tsml:heading-1>Date</tsml:heading-1>
```

```
<tsml:heading-2>field</tsml:heading-2>
<tsml:heading-3 /></tsml:header>
<tsml:header name="CHR01">
<tsml:heading-1>DBCS Char</tsml:heading-1>
<tsml:heading-2>field</tsml:heading-2>
<tsml:heading-3>length 10</tsml:heading-3>
</tsml:header>
</tsml:list-header>
<tsml:list-entries>
<tsml:entry>
<tsml:column name="BOOL01">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>False</tsml:sample-value>
<tsml:format>
<tsml:type>boolean</tsml:type>
<tsml:display-max-length>5</tsml:display-max-length>
<tsml:max-length>1</tsml:max-length>
</tsml:format>
<tsml:use-webllets>
<tsml:use-weblet name="std_boolean.std_boolean"
    technology-service="LANSA:JQMOBILE" />
<tsml:use-weblet name="std_boolean.std_boolean"
    technology-service="LANSA:XHTML" />
</tsml:use-webllets>
</tsml:column>
<tsml:column name="DAT01">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>1/01/1900</tsml:sample-value>
<tsml:format>
<tsml:type>date</tsml:type>
<tsml:display-max-length>10</tsml:display-max-length>
<tsml:max-length>10</tsml:max-length>
<tsml:sql-nullable>>true</tsml:sql-nullable>
</tsml:format>
<tsml:use-webllets>
<tsml:use-weblet name="std_input.std_input"
    technology-service="LANSA:JQMOBILE" />
<tsml:use-weblet name="std_datepicker.std_datepicker"
    technology-service="LANSA:XHTML" />
```

```
</tsml:use-weblets>
</tsml:column>
<tsml:column name="CHR01">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHJIJ</tsml:sample-value>
<tsml:format>
<tsml:type>char</tsml:type>
<tsml:display-max-length>10</tsml:display-max-length>
<tsml:max-length>10</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>J</tsml:keyboardshift>
</tsml:format>
<tsml:use-weblets>
<tsml:use-weblet name="std_input.std_input"
    technology-service="LANSA:JQMOBILE" />
<tsml:use-weblet name="std_char.std_char"
    technology-service="LANSA:XHTML" />
</tsml:use-weblets>
</tsml:column>
</tsml:entry>
</tsml:list-entries>
</tsml:list>
</tsml:lists>
</tsml:data>
```

## List Drag and Drop Output

```
<?xml version="1.0" encoding="UTF-8"?>
<lansa_design:payload xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
xmlns:lansa_design="http://www.lansa.com/2002/XML/Design"
xmlns="http://www.w3.org/1999/xhtml">
<lansa_design:imports>
<xsl:import href="std_boolean.xsl" />
<xsl:import href="std_char.xsl" />
<xsl:import href="std_datepicker.xsl" />
</lansa_design:imports>
<lansa_design:content>
<lansa_design:reference>
<xsl:apply-templates
select="/lxml:data/lxml:lists/lxml:list[@name='LIST01']"
wd:listname="LIST01">
<xsl:with-param name="allowSort" select="true()" />
<xsl:with-param name="allowColResize" select="true()" />
<xsl:with-param name="hoverEffect" select="false()" />
<xsl:with-param name="selectableRows" select="false()" />
<xsl:with-param name="hide_header_if_empty" select="true()" />
</xsl:apply-templates>
</lansa_design:reference>
<lansa_design:implementation>

<xsl:template match="/lxml:data/lxml:lists/lxml:list[@name='LIST01']">
<xsl:param name="allowSort" wd:type="std:boolean" select="true()"
wd:tip_id="" />
<xsl:param name="allowColResize" wd:type="std:boolean" select="true()"
wd:tip_id="" />
<xsl:param name="hoverEffect" wd:type="std:boolean" select="false()"
wd:tip_id="" />
<xsl:param name="selectableRows" wd:type="std:boolean" select="false()"
wd:tip_id="" />
<xsl:param name="hide_header_if_empty" wd:type="std:boolean"
select="true()" wd:tip_id="" />
```

```

<xsl:variable name="thelist"
select="/lxml:data/lxml:lists/lxml:list[@name='LIST01']" />
<input type="hidden" name="LIST01.." value="{count(lxml:list-
entries/lxml:entry[1])}" />
<div class="std_grid_wrapper" id="LIST01_wrap">
<xsl:if test="$lweb_design_mode">
<xsl:attribute name="class">std_grid_wrapper_designtime</xsl:attribute>
</xsl:if>
<table class="std_grid ui-widget" id="LIST01">
<xsl:if test="not($hide_header_if_empty) or ($thelist/@row-count != 0)">
<thead>
<tr class="list-h ui-widget-header">
<th class="ltext BOOL01 std_grid_sort_indicator" __decimalseparator=""
__formattype="boolean" __mode="input" __allowsort="true">
<xsl:for-each select="$thelist/lxml:list-header/lxml:header[1]/*[./text()
[normalize-space(.)!='']]" wd:edit-as-list="false">
<xsl:value-of select="." /><xsl:if test="not(position() = last())"><br />
</xsl:if>
</xsl:for-each>
<div class="std_grid_cell_sizer">
<xsl:if test="boolean(/lxml:data/lxml:context[@design])">
<xsl:attribute name="class">hidden__</xsl:attribute>
</xsl:if>
<xsl:comment>.</xsl:comment>
</div>
</th>
<th class="number DAT01 std_grid_sort_indicator" __decimalseparator=""
__formattype="date" __mode="input" __allowsort="true">
<xsl:for-each select="$thelist/lxml:list-header/lxml:header[2]/*[./text()
[normalize-space(.)!='']]" wd:edit-as-list="false">
<xsl:value-of select="." /><xsl:if test="not(position() = last())"><br />
</xsl:if>
</xsl:for-each>
<div class="std_grid_cell_sizer">
<xsl:if test="boolean(/lxml:data/lxml:context[@design])">
<xsl:attribute name="class">hidden__</xsl:attribute>
</xsl:if>
<xsl:comment>.</xsl:comment>
</div>

```

```

</th>
<th class="utext CHR01 std_grid_sort_indicator" __decimalseparator=""
__formattype="char" __mode="input" __allowsort="true">
<xsl:for-each select="$thelist/lxml:list-header/lxml:header[3]/*[./text()
[normalize-space(.)!="]]" wd:edit-as-list="false">
<xsl:value-of select="." /><xsl:if test="not(position() = last())"><br />
</xsl:if>
</xsl:for-each>
<div class="std_grid_cell_sizer">
<xsl:if test="boolean(/lxml:data/lxml:context[@design])">
<xsl:attribute name="class">hidden__</xsl:attribute>
</xsl:if>
<xsl:comment>.</xsl:comment>
</div>
</th>
</tr>
</thead>
</xsl:if>
<tbody class="ui-widget-content">
<xsl:for-each select="$thelist/lxml:list-entries/lxml:entry">
<xsl:variable name="BOOL01" select="lxml:column[1]" />
<xsl:variable name="DAT01" select="lxml:column[2]" />
<xsl:variable name="CHR01" select="lxml:column[3]" />
<tr __oddr="list-o" __evenrc="list-e">
<xsl:attribute name="class">
<xsl:choose>
<xsl:when test="position() mod 2">list-o</xsl:when>
<xsl:otherwise>list-e</xsl:otherwise>
</xsl:choose>
</xsl:attribute>
<td class="BOOL01">
<xsl:attribute name="__cellvalue"><xsl:value-of select="$BOOL01" />
</xsl:attribute>
<xsl:call-template name="std_boolean">
<xsl:with-param name="name" select="$BOOL01/@id" />
<xsl:with-param name="value" select="$BOOL01" />
<xsl:with-param name="display_mode" select=""input"" />
</xsl:call-template>
</td>

```

```

<td class="DAT01">
<xsl:attribute name="__cellvalue"><xsl:value-of select="$DAT01" />
</xsl:attribute>
<xsl:call-template name="std_datepicker">
<xsl:with-param name="name" select="$DAT01/@id" />
<xsl:with-param name="value" select="$DAT01" />
<xsl:with-param name="allow_sqlnull" select="true()" />
<xsl:with-param name="display_mode" select=""input"" />
<xsl:with-param name="size" select="10" />
</xsl:call-template>
</td>
<td class="CHR01">
<xsl:attribute name="__cellvalue"><xsl:value-of select="$CHR01" />
</xsl:attribute>
<xsl:call-template name="std_char">
<xsl:with-param name="name" select="$CHR01/@id" />
<xsl:with-param name="value" select="$CHR01" />
<xsl:with-param name="class" select=""utext"" />
<xsl:with-param name="display_length" select="10" />
<xsl:with-param name="display_mode" select=""input"" />
<xsl:with-param name="keyboard_shift" select=""J"" />
<xsl:with-param name="maxlength" select="10" />
<xsl:with-param name="type" select=""text"" />
</xsl:call-template>
</td>
</tr>
</xsl:for-each>
</tbody>
</table>
</div>
<script type="text/javascript">
<xsl:text disable-output-escaping="yes">//&lt;![CDATA[</xsl:text>
register_std_grid('LIST01',{
  columns: 3,
  allowSort: <xsl:value-of select="$allowSort" />,
  allowColResize: <xsl:value-of select="$allowColResize" />,
  hoverEffect: <xsl:value-of select="$hoverEffect" />,
  selectableRows: <xsl:value-of select="$selectableRows" />
});

```

```
<xsl:text disable-output-escaping="yes">]]&gt;</xsl:text>  
</script>  
</xsl:template>  
</lansa_design:implementation>  
</lansa_design:content>  
</lansa_design:payload>
```

## **Dragging and Dropping a List Column**

[List Column TSML Node](#)

[List Column Drag and Drop Output](#)

## List Column TSML Node

```
<?xml version="1.0" encoding="UTF-8"?>
<tsml:data full-document="false"
  inline="none"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-Metadata">

<tsml:weblets>
<tsml:weblet name="std_input.std_input">
<tsml:technology-services>
<tsml:technology-service name="LANSA:JQMOBILE"
  mod-id="20130102151636000">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>displayMode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>label</tsml:param-name>
<tsml:param-role>std:field_caption</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>maxlength</tsml:param-name>
<tsml:param-role>std:field_maxlength</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>rdmlxDataType</tsml:param-name>
<tsml:param-role>std:rdmlx_data_type</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>size</tsml:param-name>
<tsml:param-role>std:field_display_length</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>type</tsml:param-name>
<tsml:param-role>std:field_input_type</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
```

```
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
<tsml:weblet name="std_char.std_char">
<tsml:technology-services>
<tsml:technology-service name="LANSA:XHTML"
    mod-id="20121220115335000">
<tsml:template-params>
<tsml:template-param>
<tsml:param-name>class</tsml:param-name>
<tsml:param-role>std:field_css_class</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>display_length</tsml:param-name>
<tsml:param-role>std:field_display_length</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>display_mode</tsml:param-name>
<tsml:param-role>std:display_mode</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>height</tsml:param-name>
<tsml:param-role>std:height_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>keyboard_shift</tsml:param-name>
<tsml:param-role>std:keyboard_shift</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>maxlength</tsml:param-name>
<tsml:param-role>std:field_maxlength</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>pos_absolute</tsml:param-name>
<tsml:param-role>std:pos_absolute_design</tsml:param-role>
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>type</tsml:param-name>
<tsml:param-role>std:field_input_type</tsml:param-role>
```

```
</tsml:template-param>
<tsml:template-param>
<tsml:param-name>width</tsml:param-name>
<tsml:param-role>std:width_design</tsml:param-role>
</tsml:template-param>
</tsml:template-params>
</tsml:technology-service>
</tsml:technology-services>
</tsml:weblet>
</tsml:weblets>
```

```
<tsml:lists
  default-sample-size="5"
  column-only="true">
<tsml:list name="LIST01" inline="false">
<tsml:mode>input</tsml:mode>
<tsml:list-entries>
<tsml:entry>
<tsml:column name="CHR01">
<tsml:mode>input</tsml:mode>
<tsml:sample-value>ABCDEFGHIJ</tsml:sample-value>
<tsml:format>
<tsml:type>char</tsml:type>
<tsml:display-max-length>10</tsml:display-max-length>
<tsml:max-length>10</tsml:max-length>
<tsml:input-case>uppercase</tsml:input-case>
<tsml:keyboardshift>J</tsml:keyboardshift>
</tsml:format>
<tsml:use-weblets>
<tsml:use-weblet name="std_input.std_input"
  technology-service="LANSA:JQMOBILE" />
<tsml:use-weblet name="std_char.std_char"
  technology-service="LANSA:XHTML" />
</tsml:use-weblets>
</tsml:column>
</tsml:entry>
</tsml:list-entries>
</tsml:list>
</tsml:lists>
```

</tsml:data>

## List Column Drag and Drop Output

```
<?xml version="1.0" encoding="UTF-8"?>
<lansa_design:payload xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
xmlns:lansa_design="http://www.lansa.com/2002/XML/Design"
xmlns="http://www.w3.org/1999/xhtml">
<lansa_design:imports>
<xsl:import href="std_char.xsl" />
</lansa_design:imports>
<lansa_design:content>
<lansa_design:value>
<xsl:call-template name="std_char">
<xsl:with-param name="name" select="$CHR01/@id" />
<xsl:with-param name="value" select="$CHR01" />
<xsl:with-param name="class" select=""utext"" />
<xsl:with-param name="display_length" select="10" />
<xsl:with-param name="display_mode" select=""input"" />
<xsl:with-param name="keyboard_shift" select=""J"" />
<xsl:with-param name="maxlength" select="10" />
<xsl:with-param name="type" select=""text"" />
</xsl:call-template>
</lansa_design:value>
</lansa_design:content>
</lansa_design:payload>
```

## 7.5 Default Weblet for Technology Service

You can nominate a default weblet for a Technology Service. The default weblet is used when a field has no weblet visualization. If no default weblet is nominated, your WebRoutine XSL stylesheet should have templates to handle non-weblet fields or list columns.

The nominated weblet must exist in your Technology Service. Add the following instructions to the same level as the `<xsl:output>` instruction in your WebRoutine XSL stylesheet. This example shows the default weblet for Technology Service JQMOBILE:

```
<xsl:output method="xml" omit-xml-declaration="no" encoding="UTF-8"
indent="yes"/>
<xsl:namespace-alias stylesheet-prefix="xslt" result-prefix="xsl"/>

<!-- Default Weblet -->
<tsml:definition>
  <tsml:default-weblet name="std_input.std_input" />
</tsml:definition>
```

## **7.6 About Weblets and Weblet Templates**

[7.6.1 What are weblets and weblet templates?](#)

[7.6.2 Field weblet visualization](#)

## 7.6.1 What are weblots and weblet templates?

WAMs are shipped with a set of standard weblots for each Technology Service. A weblet is a repository object that contains one or more weblet templates. A weblet template is a reusable component that wraps some common functionality and can be dragged and dropped onto your WebRoutine designs.

Although a weblet may contain many weblet templates, it is normal practice for each weblet to contain only one template so the term "weblet" is often used to refer to a weblet template. A weblet may exist in one or more of the Technology Services.

**Note:** Weblots sometimes contain a second weblet template where the extra template is a special version of the main template for use on inline lists. These "inline" weblet templates do not show up in the Weblet Templates section of the repository because the WAM Editor will automatically use the correct template as required.

You should never modify the shipped weblots directly. Every time a Partition Initialization with the Enable for the Web is executed, the shipped weblots are re-imported and the weblots in the repository are overwritten. If you wish to customize a weblet, make a copy of the shipped weblet then modify the copy (including JavaScript functions that the weblet uses as they might be changed in future versions).

The shipped weblots use a standard naming convention where the weblet name is prefixed with 'std\_'. You should not use this prefix for any custom weblots you create.

### Also see

[Weblots for XHTML Technology Service](#)

[Weblots for jQuery Mobile Technology Service](#)

## 7.6.2 Field weblet visualization

You can define the weblet visualization of a field so that when a field is included in a web\_map, the generated page automatically uses the weblet defined in field visualization. You can define the weblet visualization for each Technology Service:

```
Begin_Com Role(*Weblet 'std_datetimepicker.std_datetimepicker')
Name(#WebletTemplate) Defaultweblet(True)
End_Com
Begin_Com Role(*Weblet 'std_mobiscroll.std_mobiscroll')
Name(#WebletTemplate2) Defaultweblet(LANSA.JQMOBILE)
End_Com
```

You can further customize your field weblet visualization without the need to create custom weblets for different fields by defining default properties for the weblets. This way, you can create weblets that can be customized via properties instead of creating separate weblets:

```
Begin_Com Role(*Weblet 'std_input.std_input')
Name(#WebletTemplateJQMOBILE) Defaultweblet(LANSA.JQMOBILE)
Default_Type("email")
End_Com
```

## 8. Weblets for XHTML Technology Service

This section documents the Weblet Templates that are shipped with the XHTML Technology Service.

The Weblet Templates repository view displays templates by group. A weblet template may belong to more than one group (right-click a weblet to configure). The standard groups are:

[Standard Weblets](#)

[Charting Weblets](#)

[Standard Field Visualizations](#)

[Layout Weblets](#)

And if you are an existing user and need to refer to them, there's [Deprecated Weblets](#).

## 8.1 Standard Weblets

This section provides a description of the standard weblets, their properties and how to use them in your own webrouines. You will not use every property available for a weblet.

<b>Weblet name</b>	<b>Description</b>
<a href="#">Anchor (std_anchor)</a>	The anchor weblet provides a hyperlink (or anchor) control.
<a href="#">Attachment panel (std_attach_panel_v2)</a>	Provides a panel with five areas where content can be dropped: left, top, right, center, and bottom.
<a href="#">Autocomplete (std_autocomplete)</a>	While you type, the Autocomplete weblet provides suggestions provided by a WebRoutine using Ajax.
<a href="#">Checkbox (std_checkbox)</a>	Checkbox with a caption.
<a href="#">CKEditor (std_ckeditor)</a>	CKEditor is a WYSIWYG rich text editor.
<a href="#">Clickable image (std_click_image)</a>	Image that can be clicked on.
<a href="#">Combo box (std_dropdown)</a>	Drop down items can be specified via items property in the LANSA Editor or they can come from a list.
<a href="#">Dynamic Select Box (std_dynamic_select)</a>	An element that allows you to create a dropdown or a list that can monitor another field and automatically refresh itself when that field changes.
<a href="#">Export to Excel (std_toexcel)</a>	Allows you to export a table or grid to an Excel spreadsheet.
<a href="#">File Upload (std_fileupload)</a>	Allows you to select files to upload to the application server. The webrouine that receives the uploaded files can manipulate them as required.
<a href="#">Grid (std_grid_v2 and std_grid_v3)</a>	Grid control with sortable columns. Grid cells can be populated from a list.

Image (std_image)	Image with the option to delay loading until the image comes into view.
Horizontal splitter (std_splitter_horz)	Horizontal splitter control that allows content to be added for top and bottom splitter panes.
Large List (std_largelist)	Large list. Used for report like (output only) lists with very simple formatting. List can be sent as XHTML or CSV (Comma-separated values).
Listbox (std_listbox)	List box control. The items can come from a list or specified directly through items property.
List paging buttons (std_list_buttons)	Buttons that can be used for browse list navigation.
List paging images (std_list_images)	Image buttons that can be used for browse list navigation.
Mark-up (std_markup)	Used when you want to visualize the content in output mode only. Companion to the CKEditor weblet.
Memo using a list (std_textarea_v2)	A multi-line edit control where each line is loaded and stored to a list.
Memo using a field (std_list_textarea)	A multi-line edit control using a field. Superseded by std_textarea_v2.
Menu bar (std_menubar)	Provides the functionality of a menu bar that can invoke other web pages including other web routines.
Menu item (std_menu_item_v2)	A hyperlink menu item. Used when creating HTML menus.
Messages (std_messages)	Shows messages from a WEBROUTINE that have been output with MESSAGE RDML command.
Navigation panel (std_nav_panel)	A panel that can navigate to a WEBROUTINE or a URL independently of the rest of the page.
Panel (std_panel)	A panel that allows contents such as html or other weblets to be dropped on it. All of its content is

relatively positioned in the panel. It also has "snap to grid" design time support.

[Print Page](#)  
(std\_printpage)

Provides a hyperlink to print the current page.

[Progress bar](#)  
(std\_progressbar)

Display status of a determinate or indeterminate process.

[Prompter](#)  
(std\_prompter)

A button that supports field prompting. The prompter can invoke a WEBROUTINE, even from a different WAM, for its pop up window page.

[Push Button](#)  
(std\_button\_v2) &  
[Push Button with Images](#)  
(std\_image\_button\_v2)

These weblets provide themable push buttons for your web page.

The previous versions of these Weblets have been deprecated.

[Radio button](#)  
(std\_rad\_button)

A radio button.

[Radio group](#)  
(std\_radbuttons)

A radio button group.

[Tab pages](#)  
(std\_tab\_pages\_v2)

Tab control that allows content to be added for each tab page. Tab pages can be changed, added and deleted. You can modify tab captions through the tabs property.

[TreeView](#)  
(std\_treeview\_v2)

Provides an expandable collapsible tree

[Vertical splitter](#)  
(std\_splitter\_vert)

Vertical splitter control that allows content to be added for left and right splitter panes.

## 8.1.1 Anchor (std\_anchor)

### QuickStart - Anchor Properties - Anchor

The anchor weblet provides a hyperlink (or anchor) control. It broadly corresponds to the <a> (anchor) HTML element that designates the destination of a hypertext link.

- The anchor weblet can display an image and/or text to represent the link and can specify a destination that the WAM should navigate to when the link is activated.
- The image or text can be static (specified as literals in the weblet properties) or can be determined by nominated fields, system variables or multilingual variables.
- The destination can be a url (such as <http://lansa//www.yourcompany.com/>) or you can specify a WAM and webroutine to be executed and optionally identify a field whose value should be passed to the webroutine.

The anchor weblet looks like this (the department codes):

Dept Code	Department Description
<a href="#">ADM</a>	ADMINISTRATOR DEPT
<a href="#">AUD</a>	INTERNAL AUDITING
<a href="#">FLT</a>	FLEET ADMINISTRATION

The anchor weblet is frequently used with a field or a column in a list to provide a quick and easy way to both select an item and initiate an action concerning that item. In the example above, the department codes have been made into anchors. When the user clicks a department code another webroutine is invoked that displays details for the corresponding department. The `currentrowhfield` and `currentrownumval` properties specify that the corresponding department code is passed to that webroutine.

## QuickStart - Anchor

To use an anchor with a column in a list, you would need to create a webroutine that specifies the list in its WEB\_MAP as \*BOTH or \*OUTPUT. When you open the generated XSL in the LANSAs Editor, you can change a column of the list to function as an anchor as follows:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Anchor weblet.
2. Drag the Anchor weblet over the column in your list and release the left-mouse button. The column's appearance changes to show that it is now an anchor. Click on an item in the column and then click on the Details tab. You should see that the name and value properties for the anchor weblet have already been set according to the field upon which it was dropped.
3. Set the currentrowhfield and currentrownumval properties as described in the property descriptions.
4. Set the on\_click\_wrname property to the name of the webroutine to be invoked when the hyperlink is clicked. If the webroutine is in a different WAM to the current webroutine then you will need to set the on\_click\_wamname property as well.

## Properties - Anchor

The Anchor weblet's properties are:

absolute-image-path	on_click_wname	show_in_new_window
currentrowhfield	pos_absolute_design	tab_index
currentrownumval	presubmit_js	target_window_name
formname	protocol	text_class
hide_if	reentryfield	url
mouseover_class	reentryvalue	value
name	relative-image-path	vf_wamevent
on_click_wamname		width_design

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

This property specifies the text that is displayed for the hyperlink. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – you must specify the value if you wish the hyperlink to display text (the hyperlink may also display an image – see the `relative-image-path` and `absolute-image-path` properties).

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **currentrowhfield**

The field name to be used to post to the target webroutine the value that is specified in the currentrownumval property. The field name should be in single quotes.

See the description of the currentrownumval property for further information.

### **Default value**

'STDROWNUM'

### **Valid values**

Single-quoted text.

### **Example**

This example specifies the field name DEPTLINK as the field name to be used to post the value to the target webroutine. The target webroutine would need to have field DEPTLINK in its WEB\_MAP for \*BOTH or for \*INPUT in order to receive the value:

 currentrowhfield	'DEPTLINK'
--	------------

## **currentrownumval**

The value to post to the target webroutine in the field specified in the `currentrowhfield` property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the `currentrowhfield` property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. `currentrowhfield`: the field name that the target webroutine uses to refer to the information
2. `currentrownumval`: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** Despite the name of the property being *currentrownumval*, the field name specified in `currentrownumval` is not required to be a numeric field.

### **Default value**

`position()`

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

See the description of the reentryvalue property for further information.

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

'STDREENTRY'

### **Valid values**

Single-quoted text.

## **reentryvalue**

The value to post to the target webroutine in the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the reentryfield property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. reentryfield: the field name that the target webroutine uses to refer to the information
2. reentryvalue: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

'D'

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

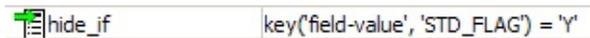
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered in this form:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is '#STD\_FLAG = 'Y''. To the left of the text is a small icon representing a document or list.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. To the left of the text is a small icon representing a document or list.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **url**

This property may be used to specify a URL that the hyperlink will navigate to. If specified, the URL may be specified as a literal value (for example 'http://www.mycompany.com/') or a field name may be specified that contains the URL at run-time.

This property takes precedence over the `on_click_wamname`, `on_click_wrname` and `protocol` properties. The latter properties are ignored if `url` is specified.

### **Default Value**

'javascript:void();' – equivalent to nothing.

### **Valid Values**

A URL enclosed by single quotes or the name of a field, system variable or multilingual variable that will contain the URL at run-time.

## **on\_click\_wamname**

Specifies the name of the WAM whose webroutine is executed when the hyperlink that represents this weblet is clicked. (The webroutine name is specified in the on\_click\_wrname property.)

This property is ignored if the url property is specified.

### **Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName)

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

Specifies the name of the webroutine that is executed when the hyperlink that represents this weblet is clicked. (The name of the WAM containing the webroutine is specified in the on\_click\_wamname property.)

This property is ignored if the url property is specified.

### **Default value**

No default value applies – either the url property or the on\_click\_wrname property must be specified

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

### **Example**

This example specifies deptdetail as the name of the webroutine that will be executed when the hyperlink is clicked. The name of the WAM containing the webroutine is specified by the on\_click\_wamname property.

 on_click_wrname	'deptdetail'
---	--------------

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

Typically you might use this property when it is necessary to switch to or from secure-mode processing. Otherwise it is not usually necessary to specify this property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. If specified, it is usually 'http:' or 'https:'.

## **show\_in\_new\_window**

A boolean property, the result of which determines whether response HTML for the weblet should be shown in a new browser window.

### **Default value**

false() – response HTML is shown in the current browser window.

### **Valid values**

true(), false() or a valid expression.

## **target\_window\_name**

The name of the window, or frame, in which the destination of the hyperlink will be shown.

### **Default value**

Blank – the destination of the hyperlink will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that Position Absolutely must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

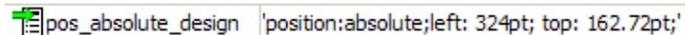
Blank (not positioned)

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSA Editor. This resulted in the value shown for the pos\_absolute\_design property.

A screenshot of a property editor window. The window has a title bar with a green icon and the text 'pos\_absolute\_design'. The main content area shows the text 'position:absolute;left: 324pt; top: 162.72pt;'.

pos_absolute_design	'position:absolute;left: 324pt; top: 162.72pt;'
---------------------	---

## **width\_design**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design property. However you can directly edit the property value if required.

### **Default value**

Blank (weblet uses its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **relative-image-path**

The path and file name, relative to the images virtual directory, of the image to be displayed.

### **Default value**

Blank – no image is displayed.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **absolute-image-path**

The path and file name of the image to be displayed. If specified, the `relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet. For example, 'std\_anchor'.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **mouseover\_class**

The Cascading Style Sheet (CSS) class name of the weblet when the mouse is moved over it.

### **Default value**

No default value applies for this weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown list in the property sheet. For example, the `mouseover_class` property after the shipped mouseover class has been selected could be `'std_anchor_mouseover'`.

**text\_class**

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

The name of the shipped text class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **presubmit\_js**

JavaScript code to be run prior to navigating to the destination of the hyperlink.  
JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

vf\_wamevent VLF WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.2 Autocomplete (std\_autocomplete)

### [QuickStart – Autocomplete Properties – Autocomplete](#)

The Autocomplete weblet provides suggestions while you type into the field. The suggestions are provided by a webroutine using Ajax:



## **QuickStart – Autocomplete**

To use the autocomplete weblet you can follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the jQuery UI Autocomplete weblet.
2. Drag and drop the weblet onto your page. Make sure the weblet is selected and then click on the Details tab. Fill the properties as required.
3. While the autocomplete weblet is selected in the designer, right click and from the context menu, select the option to create the Ajax WebRoutine.
4. In your RDMLX source, complete the code for the response webroutine (created in step 3).

## Properties – Autocomplete

The Autocomplete weblet's properties are:

cache	labelField	scrollHeight
class	listName	size
delay	matchContains	sourceWamName
disabled	maxLength	sourceWrName
display_length	minLength	tab_index
display_mode	namevalue	termField
extraFields	onchange_script	title
height	pos_absolute	valueField
hide_if	read_only	width
keyboard_shift	scroll	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box or left blank for users to fill with suggested values.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden. It will only work in autocomplete mode if the weblet is in input mode.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

**size**

The size of the weblet data in characters/bytes.

This property is currently not implemented – use the `maxlength` and/or `display_length` properties instead.

## **display\_length**

The approximate size of the weblet input box in characters – the browser sizes the input box according to the number of characters specified. If the width property is specified, it takes precedence and the display\_length property is ignored.

### **Default value**

Blank (the weblet assumes a default size).

### **Valid values**

A numeric value.

## **keyboard\_shift**

The keyboard shift for the input field.

### **Default value**

The keyboard shift of the field with this weblet visualization. Blank otherwise.

### **Valid values**

Char and String data types: ' ', 'W', 'J', 'E', 'O' and 'U'

Alpha data type: ' ', 'X', 'A', 'N', "W", 'T', 'D', 'M', 'J', 'E' and 'O'

The keyboard shift is currently only used to validate DBCS fields.
--

## **minLength**

The minimum number of characters a user has to type before the Autocomplete activates. Zero is useful for local data with just a few items. Should be increased when there are a lot of items, where a single character would match a few thousand items.

## **Default value**

1

## **Valid values**

Numeric value.

**delay**

The delay in milliseconds the Autocomplete waits after a keystroke to activate itself. A zero-delay makes sense for local data (more responsive), but can produce a lot of load for remote data, while being less responsive.

**Default value**

300 milliseconds

**Valid values**

Numeric value in milliseconds.

**sourceWamName**

The name of the WAM whose Webroutine provides the response data for this weblet.

**Default value**

The current WAM

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**sourceWrName**

The name of the Webroutine that provides the response data for this weblet.

**Default value**

Blank – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must be a JSON response weboutine and exist in the WAM specified in the sourceWamName property. A list of known JSON Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**termField**

The field name in the response handling webroutine that is to receive the current value in the autocomplete field.

**Default value**

None. Required field.

**Valid values**

Single quoted text field. Field must be in WEB\_MAP for \*INPUT in the response handling webroutine.

**listName**

[Optional] The name of the list in the response webroutine to store the suggestion list. If left empty the first list (which should be the only list in the response webroutine) will be used.

**Default value**

Blank: It uses the first list in response webroutine.

**Valid values**

Single quoted text . List must be in WEB\_MAP for \*OUTPUT in the response handling webroutine.

## **labelField**

The response data is a list, with either a label or value column or both. The label column is displayed in the suggestion menu. The value will be inserted into the input element after the user selected something from the menu. If just one column is specified, it will be used for both, eg. if you provide only value-properties, the value will also be used as the label.

## **Default value**

Blank.

## **Valid values**

Single quoted text. Column name but exist in list returned by the response handling webroutine.

## **valueField**

The response data is a list, with either a label or value column or both. The label column is displayed in the suggestion menu. The value will be inserted into the input element after the user selected something from the menu. If just one column is specified, it will be used for both, eg. if you provide only value-properties, the value will also be used as the label.

## **Default value**

Blank.

## **Valid values**

Single quoted text. Column name but exist in list returned by the response handling webroutine.

**extraFields**

[Optional]. A comma separated list of fields to send to the webroutine providing the response for this weblet.

**Default value**

Blank. No extra fields

**Valid values**

Comma separated list of fields. Fields must be input or hidden fields.

**cache**

Set to true to save the server response locally and filter it to narrow suggestions as the user types more characters.

**Default value**

True.

**Valid values**

true(), false() or a valid expression.

## **matchContains**

If set to true, autocomplete matches the entered characters anywhere in the suggestions. Otherwise, autocomplete only matches suggestions that start with the entered characters. This applies to narrowing the suggestions for cached responses. The data source must have a selection criteria consistent with this property.

### **Default value**

False.

### **Valid values**

true(), false() or a valid expression.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

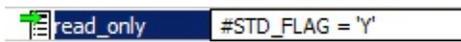
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

**scroll**

Whether to scroll when more results than configured via scrollHeight are available.

**Default value**

False.

**Valid values**

true(), false() or a valid expression.

**scrollHeight**

Height of scrolled autocomplete suggestion box. Only activated if scroll is true.

**Default value**

180px.

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## **onchange\_script**

JavaScript code bind to the jQuery UI autocomplete widget change event (when the input box loses focus after the text has been changed). It has optional event and ui object parameters. For example, `myOnChangeEventHandler(event, ui)`.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## **onselect\_script**

JavaScript code bind to the jQuery UI autocomplete widget select event (when the user selects an item from the autocomplete suggestion list). It has optional event and ui object parameters. For example, myOnSelectEventHandler(event, ui).

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

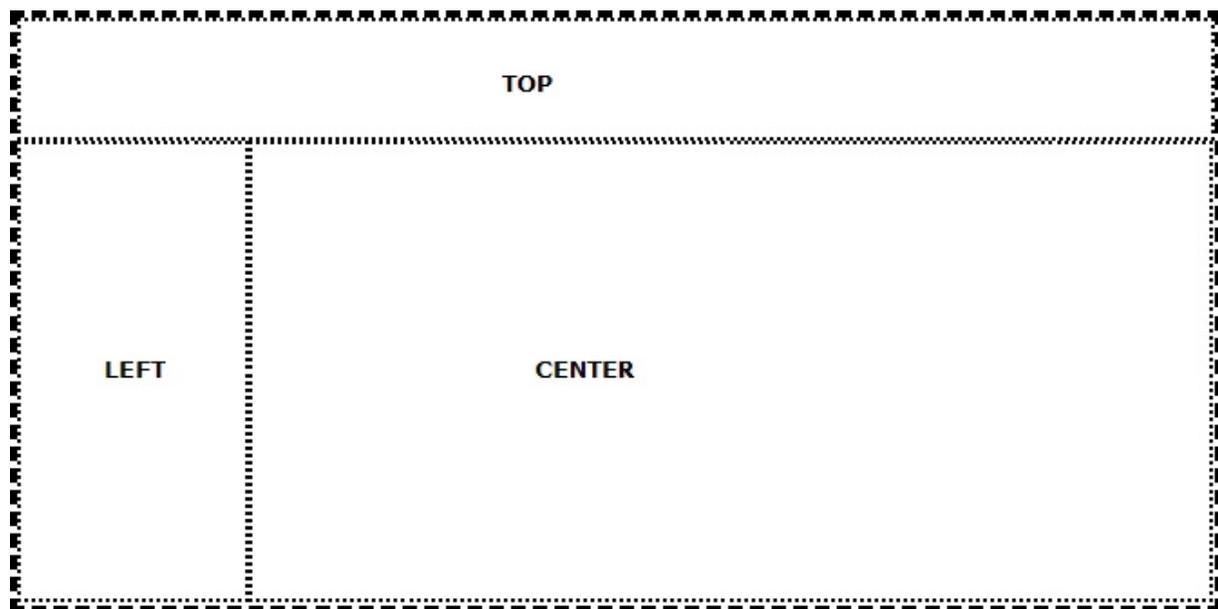
Any valid JavaScript statement(s).

### 8.1.3 Attachment panel (std\_attach\_panel\_v2)

#### [QuickStart- Attachment panel Properties - Attachment panel](#)

The Attachment panel weblet provides a panel with five areas where content can be dropped: left, top, right, center, and bottom. Each of these has attachment layout manager behavior. Contents can be inserted or other weblets dropped into any of the five areas. Dropped weblets are sized according to attachment layout manager rules when they are dropped.

The following is an example of the appearance of a nearly empty attachment panel. In this example, just three of the attachment areas have been used. A thick dashed border has been specified for the attachment panel and thin dotted borders for the panels used in the three areas. The borders have been used for clarity in this example – you do not have to use visible borders and you may not wish to in your applications. Remember you can drag and drop other weblets (such as input boxes, check boxes and push buttons) onto each of the layout areas.



The attachment panel is one of a number of weblets that you can use to aid the creation of a consistent and visually appealing layout for your web pages. You may also wish to review the horizontal and vertical splitters and the panel and navigation panel weblets. This weblet (the attachment panel) is static – the user is not able to resize or otherwise manipulate the size and position of the panels that it contains.

Version 2 of the attachment panel deals with some browser compatability issues by replacing the border and border\_width properties with a single border property and removes the class\_top, class\_left, class\_center, class\_right and class\_bottom properties. The class values can now be set using the "panes" custom property editor which also allows you to set the size and alignment of each pane.

## **QuickStart- Attachment panel**

To use the attachment panel you can follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Attachment panel weblet.
2. Drag and drop the weblet onto your page. Make sure the weblet is selected and then click on the Details tab. Set any properties required for the attachment panel, such as borders.
3. Now you can drag and drop or otherwise insert content into the required panes or layout areas. You may find it easiest to drag and drop the Panel weblet into each of the five layout areas that you wish to use. You can then more easily size those panels and insert other weblets onto those child panels.

## Properties - Attachment panel

The Attachment panel weblet's Properties are:

name	panes
border	pos_absolute
height	width
hide_if	

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## panes

An XML node set specifying a set of panes to show and their properties. This is a system generated value set up when you drag the attachment panel onto the design view. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet.

## Default value

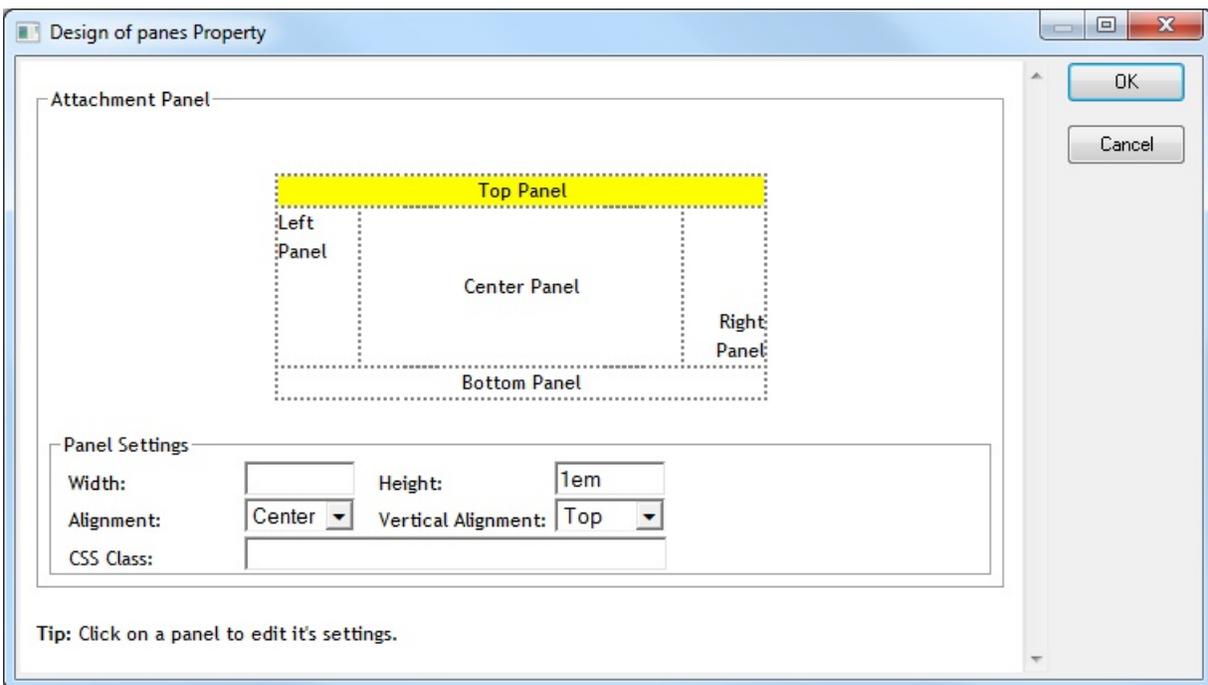
`document(")/*/lxml:data/lxml:panes[@id='<unique id>']` (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

## Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## Example

Use the ellipse button on the property to open the designer:



The designer allows the following properties to be edited for each pane:

- Width/Height: You must supply a valid CSS length value
- Vertical/Horizontal alignment: Valid values are displayed in a dropdown
- CSS Class: A CSS class to apply to the pane.

## **border**

A CSS border value for the outer boundary of the weblet. For example '1px dashed red'.

### **Default value**

Blank (no border is shown).

### **Valid values**

Any valid CSS border string. This consists of a width, a style and a color, each separated by a space. Properties can be omitted from this list but must be in this order. For example, "solid #ff0000" is valid. For more information on the CSS border property, look at the [W3C specification](#).

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSAs Editor. This resulted in the value shown for the pos\_absolute\_design property.

 pos_absolute_design	'position:absolute;left: 324pt; top: 162.72pt;'
---	---

## **width**

The width of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

" (this specifies that the attachment panel will use the full width available in the containing element).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSa Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

'250pt'

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.1.4 Push Button (std\_button\_v2) & Push Button with Images (std\_image\_button\_v2)

[QuickStart- Push Button & Push Button with Images](#)

[Properties - Push Button & Push Button with Images](#)

The Push Button weblets provide themable push buttons for your web page. They look like this:



## **QuickStart- Push Button & Push Button with Images**

To add a push button to your web page:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate either of the Push Button weblets.
2. Drag and drop the required weblet onto the web page. Click on the Details tab.
3. Set the caption to specify the text to be displayed on the button. In the case of the Push Button with Images, set the appropriate image properties. A left-hand side image and a right-hand side image can be set.
4. Set the `on_click_wrname` properties to the name of the webroutine to be invoked when the button is clicked. If the webroutine is in a different WAM to the current webroutine then you will also need to set the `on_click_wamname` property.

## Properties - Push Button & Push Button with Images

All these properties are common to both button weblets except for those indicated as *std\_image\_button\_v2 only*.

caption	left_absolute_image_path	right_image_height
currentrowhfield	left_image_height	right_relative_image_path
currentrownumval	left_relative_image_path	show_in_new_window
confirm	name	submitExtraFields
confirmText	on_click_wamname	tab_index
default_button	on_click_wrname	target_window_name
disabled	pos_absolute_design	text_class
formname	presubmit_js	title
height_design	protocol	vf_wamevent
hide_if	right_absolute_image_path	width_design

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name enclosed in single quotes.

**caption**

The caption for the weblet.

**Default value**

'Caption'

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **currentrowhfield**

The field name to be used to post to the target webroutine the value that is specified in the currentrownumval property. The field name should be in single quotes.

See the description of the currentrownumval property for further information.

### **Default value**

'STDROWNUM'

### **Valid values**

Single-quoted text.

## **currentrownumval**

The value to post to the target webroutine in the field specified in the `currentrowhfield` property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the `currentrowhfield` property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. `currentrowhfield`: the field name that the target webroutine uses to refer to the information
2. `currentrownumval`: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** Despite the name of the property being *currentrownumval*, the field name specified in `currentrownumval` is not required to be a numeric field.

### **Default value**

`position()`

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **left\_relative\_image\_path**

*std\_image\_button\_v2 only.*

The path and name, relative to the images directory, of the image to be displayed on the left of the weblet. If specified, the `left_absolute_image_path` property should be left blank.

### **Default value**

'ball\_red.gif'

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **left\_absolute\_image\_path**

*std\_image\_button\_v2 only.*

The path and name of the image to be displayed on the left of the weblet. If specified, the `left_relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `left_relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

## **left\_image\_height**

*std\_image\_button\_v2 only.*

The height of the image on the left of the weblet.

### **Default value**

'12pt'

### **Valid values**

A height, in a valid unit of measurement, enclosed in single quotes.

## **right\_relative\_image\_path**

*std\_image\_button\_v2 only.*

The path and name, relative to the images directory, of the image to be displayed on the right of the weblet. If specified, the `right_absolute_image_path` property should be left blank.

### **Default value**

Blank – by default, buttons do not display an image on the right.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **right\_absolute\_image\_path**

*std\_image\_button\_v2 only.*

The path and name of the image to be displayed on the right of the weblet. If specified, the `right_relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `right_relative_image_path` property, if specified.

### **Valid values**

The path and name of an image, enclosed in single quotes.

## **right\_image\_height**

*std\_image\_button\_v2 only.*

The height of the image on the right of the weblet.

### **Default value**

'12pt'

### **Valid values**

A height, in a valid unit of measurement, enclosed in single quotes.

## submitExtraFields

An XML nodeset specifying any extra fields (not already in the form being submitted) that should be sent to the onClick webroutine. This will most commonly be used when the weblet is used in a list or grid to specify values from other columns in the list.

### Default value

document(")/\*/lxml:data/lxml:json[not(@id)] (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

### Example

This example shows the submitExtraFields property editor:

Name	Value
DEPARTMENT	SELDEPT
SECTION	SELSECT

This shows how output fields in the current webroutine (the "Value" column) can be mapped to input fields with a different name (the "Name" column) defined in the onClick webroutine's WEB\_MAP.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**on\_click\_wamname**

The name of the WAM to be invoked when the weblet is clicked.

**Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_click\_wrname**

The name of the Webroutine to be invoked when the weblet is clicked.

**Default value**

Blank – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the on\_click\_wname property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **show\_in\_new\_window**

A Boolean property, the result of which determines whether response HTML for the weblet should be shown in a new browser window.

### **Default value**

false() – response HTML is shown in the current browser window.

### **Valid values**

true(), false() or a valid expression.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that `_search` and `_media` are supported by Internet Explorer 6 only.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

**title**

Text to be displayed as a Tool Tip for the weblet when the mouse is hovered over it.

**Default value**

Blank – no Tool Tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**text\_class**

The Cascading Style Sheet class name for the caption in the push button.

**Default value**

None for the push button. 'std\_image\_button\_text' for the image push button.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **presubmit\_js**

JavaScript code to be run prior to the submission of the form.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript function, or JavaScript code followed by a semicolon (;).

If you want to execute the presubmit JavaScript only, without running the JavaScript that submits the request (thus canceling the onclick event), append **return false;** to your presubmit JavaScript.

## **confirm**

If true, it presents a confirmation dialog box for the user to confirm the action before proceeding. If the user clicks on the Cancel button in the confirmation dialog box or closes the confirmation dialog box without clicking on the OK button, the action to be performed by the button weblet is cancelled.

## **Default value**

false() –No confirmation dialog box.

## **Valid values**

true(), false() or a valid expression.

## **Example**

This example shows the confirmation dialog box you would add to a Delete button:



**confirmText**

The text to display in the confirmation dialog box. Only applicable if the confirm property is set to true().

**Default value**

Blank – no text message.

**Valid values**

Single-quoted text, the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list) or an XPath expression whose result is a string.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **default\_button**

A Boolean property, the result of which determines whether the button is the default button for the form. Only one button on the form can be the default button – setting to True will set all other buttons to False.

## **Default value**

Blank - Equivalent to False.

## **Valid values**

true(), false() or a valid expression.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.5 Checkbox (std\_checkbox)

### [QuickStart - Checkbox Properties - Checkbox](#)

The checkbox weblet provides a checkbox control. It broadly corresponds to the `<input type="checkbox">` HTML element.

A checkbox control is typically used to represent a value that can have one of two states. For example: on or off; yes or no; selected or unselected.

When used in a list (with no caption), the checkbox weblet looks like this:

Dept Code	Department Description
<input type="checkbox"/>	ADM ADMINISTRATOR DEPT
<input checked="" type="checkbox"/>	AUD INTERNAL AUDITING
<input type="checkbox"/>	FLT FLEET ADMINISTRATION

**Note:** While the checkbox weblet includes properties such as `on_click_wname` that allow it to navigate to another webroutine when clicked, It is not good user-interface design practice to initiate actions from the click of a checkbox. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

## QuickStart - Checkbox

It is common to use a checkbox with a column in a list to indicate a data item or selection state. To do so you would need to create a webroutine that specifies the list in its WEB\_MAP. When you open the generated XSL in the LANSAs Editor, you can change a column of the list to function as a checkbox as follows: Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Checkbox weblet.

Drag the Checkbox weblet over the column in your list and release the left-mouse button. The column's appearance changes to show that it is now a checkbox. Click on an item in the column and then click on the Details tab. You should see that the name and value properties for the checkbox weblet have already been set according to the field upon which it was dropped.

Set the caption property as required. When used in a list you may not wish to use a caption – to remove the caption, specify an empty string by specifying two quote marks with no contained text.

You may need to set the oncode and offcode properties according to the values used in your application to represent the on and off or selected and unselected states.

## Properties - Checkbox

The Checkbox weblet's Properties are:

alignment	mouseover_class	protocol
caption	name	reentryfield
class	offcode	reentryvalue
disabled	on_click_wamname	tab_index
display_mode	on_click_wrname	target_window_name
formname	oncode	text_class
hide_if	pos_absolute	value
		vf_wamevent

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the checkbox and/or that is used to receive the state of the checkbox.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**caption**

The caption for the weblet. The caption is displayed adjacent to the checkbox on the web page.

**Default value**

'Caption' (this is a placeholder default value - you will need to set a caption).

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

If you do not want any caption displayed next to the checkbox (for example, when it is used in a list) you may specify an empty string by specifying two quote marks with no contained text.

**oncode**

The value that represents, or is used to set, a checked status for a checkbox.

**Default value**

'Y'

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**offcode**

The value that represents, or is used to set, an unchecked status for a checkbox.

**Default value**

'N'

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

See the description of the reentryvalue property for further information.

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

'STDREENTRY'

### **Valid values**

Single-quoted text.

## **reentryvalue**

The value to post to the target webroutine in the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the reentryfield property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. reentryfield: the field name that the target webroutine uses to refer to the information
2. reentryvalue: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

'M'

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

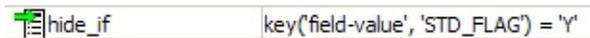
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value '#STD\_FLAG = 'Y'' is entered in a text field.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is shown in a text field. To its right, the value 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in another text field.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **on\_click\_wamname**

Specifies the name of the WAM whose webroutine is executed when the checkbox is clicked. (The webroutine name is specified in the on\_click\_wrname property.) This property is ignored unless the on\_click\_wrname property is specified.

**Note:** It is not good user-interface design practice to initiate actions from the click of a checkbox. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

Specifies the name of the webroutine that is executed when the checkbox is clicked. (The name of the WAM containing the webroutine is specified in the on\_click\_wamname property.)

**Note:** It is not good user-interface design practice to initiate actions from the click of a checkbox. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

No default value applies.

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

Typically you might use this property when it is necessary to switch to or from secure-mode processing. Otherwise it is not usually necessary to specify this property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. If specified, it is usually 'http:' or 'https:'.

## **target\_window\_name**

The name of the window, or frame, in which the response HTML will be shown upon navigation to the webroutine specified in the on\_click\_wname property.

### **Default value**

Blank – the response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

**alignment**

Determines whether the caption appears to the left or right of the checkbox.

**Default value**

'right'

**Valid values**

'left', 'right' or the name of a field or system variable that will contain one of the preceding values at run-time.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSAs Editor. This resulted in the value shown for the pos\_absolute\_design property.

 pos_absolute_design	'position:absolute;left: 324pt; top: 162.72pt;'
---	---

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**mouseover\_class**

The Cascading Style Sheet (CSS) class name of the weblet when the mouse is moved over it.

**Default value**

No default value applies for this weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**text\_class**

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

The name of the shipped text class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

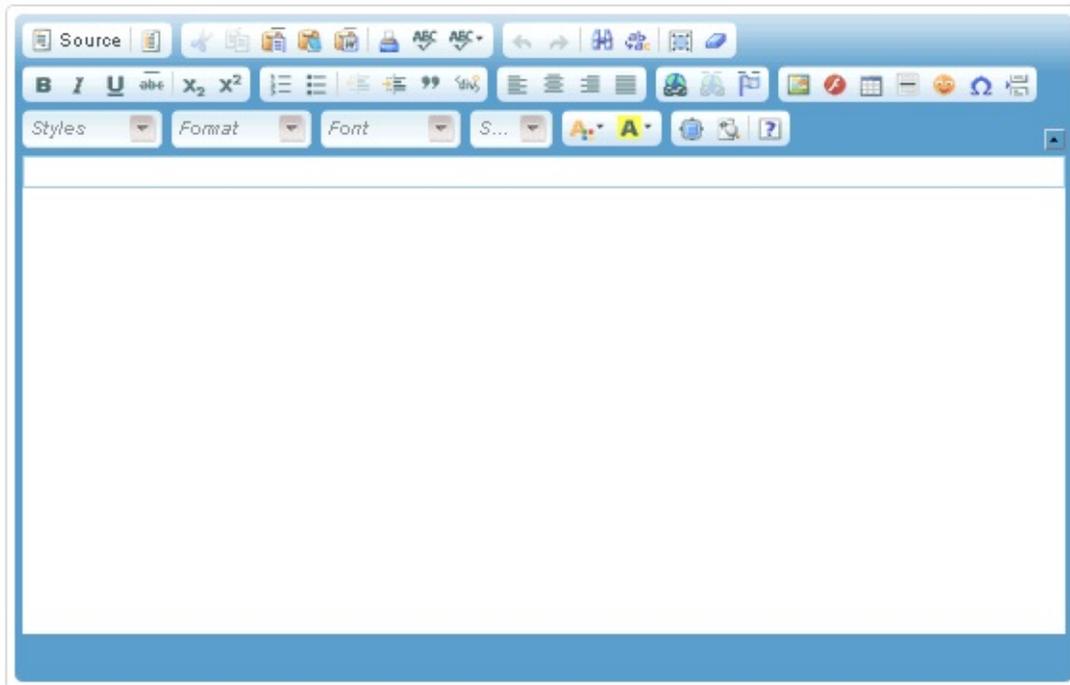
**Valid values**

String value. Comma (',') not allowed.

## 8.1.6 CKEditor Rich Text Editor (std\_ckeditor)

[QuickStart – CKEditor](#) [Properties – CKEditor Rich Text Editor](#)

CKEditor is a rich text editor. It's a WYSIWYG editor, which means that the text being edited on it looks as similar as possible to the results users have when publishing it.



## **QuickStart – CKEditor**

To use the CKEditor weblet you can follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the CKEditor Rich Text Editor weblet.
2. Drag and drop the weblet onto your page. Make sure the weblet is selected and then click on the Details tab.
3. Assign a field name to the CKEditor name property. The field would normally be a string field long enough to store the expected content when escaped (all markup text is escaped by the CKEditor).

## Properties – CKEditor Rich Text Editor

autoGrow	pos_absolute	showSource
autoGrow_maxHeight	resize_dir	tab_index
autoGrow_minHeight	resize_enabled	toolbar
contentCss	resize_maxHeight	toolbarCanCollapse
height_design	resize_maxWidth	uiColor
hide_if	resize_minHeight	value
name	resize_minWidth	valueFromField
onchange_script	showElementsPath	width_design

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name.

Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **valueFromField**

If set to true and the value property is null, the CKEditor weblet will load the value from the field that matches the CKEditor name attribute. Use this option if your text content is large and you don't want the content to appear both in the CKEditor weblet value and the webroutine field values list.

### **Default value**

False()

### **Valid values**

Any valid XPath expression that returns a Boolean value.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## toolbar

The level of features to include in the editor. The Full option shows all the options available in the CKEditor. Use the basic toolbar if you want to provide simple editing capabilities only.

## Default value

Basic

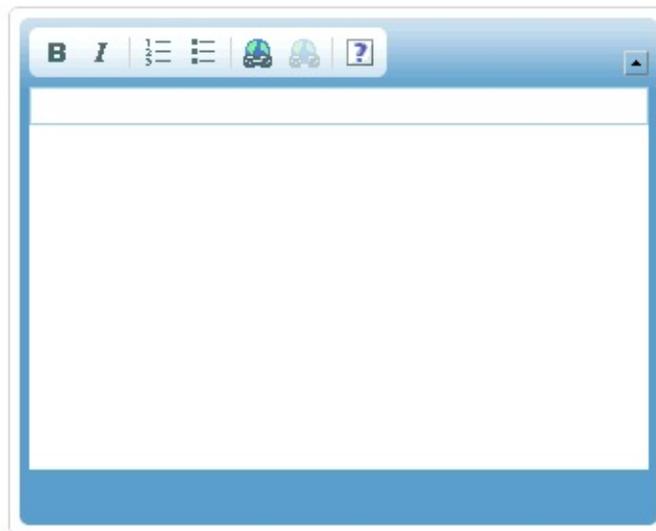
## Valid values

Basic

Full

## Example

CKEditor with basic toolbar:



## **showSource**

When showing the full toolbar, whether to allow users to view/edit the HTML source

Allowing users to add any markup in field values could be a security vulnerability. Only use this option with care.

### **Default value**

False.

### **Valid values**

False(), true() or any valid XPath expression that returns a Boolean value.

## **showElementsPath**

The elements path displays information about the HTML elements of the document for a position of the cursor. It appears in the status bar of the CKEditor.

### **Default value**

True.

### **Valid values**

False(), true() or any valid XPath expression that returns a Boolean value.

## **toolbarCanCollapse**

Whether the toolbar can be collapsed by the user. If disabled, the collapse button will not be displayed.

### **Default value**

True.

### **Valid values**

False(), true() or any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width\_design**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property value if required.

### **Default value**

Blank.

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height\_design**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

Blank.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **resize\_enabled**

Whether to enable the CKEditor to be resized. If disabled the resize handler will not be visible.

### **Default value**

False.

### **Valid values**

False(), true() or any valid XPath expression that returns a Boolean value.

**resize\_dir**

The directions to which the editor resizing is enabled. Only applicable if the CKEditor is enabled to be resized.

**Default value**

Both

**Valid values**

Both

Vertical

Horizontal

**autoGrow**

Whether to enable AutoGrow. Autogrow allows the content area to expand as the user fills the content area.

**Default value**

False.

**Valid values**

False(), true() or any valid XPath expression that returns a Boolean value.

## **autoGrow\_maxHeight**

The maximum height to which the editor can reach using AutoGrow.

### **Default value**

Blank.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **autoGrow\_minHeight**

The minimum height to which the editor can reach using AutoGrow.

### **Default value**

Blank.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **resize\_maxHeight**

The maximum editor height, in pixels, when resizing it with the resize handle.

### **Default value**

3000 pixels

### **Valid values**

A height in pixels.

## **resize\_maxWidth**

The maximum editor width, in pixels, when resizing it with the resize handle.

### **Default value**

3000 pixels

### **Valid values**

A width in pixels.

## **resize\_minHeight**

The minimum editor height, in pixels, when resizing it with the resize handle.

Note: It fallbacks to editor's actual height if that's smaller than the default value.

### **Default value**

250 pixels

### **Valid values**

A height in pixels.

## **resize\_minWidth**

The minimum editor width, in pixels, when resizing it with the resize handle.

Note: It fallbacks to editor's actual width if that's smaller than the default value.

### **Default value**

750 pixels

### **Valid values**

A width in pixels.

## **contentCss**

The CSS file(s) to be used to apply style to the contents (comma separated list). It should reflect the CSS used in the final pages where the contents are to be used. Use the special value 'inherit' to apply the same CSS used in the webroutine.

### **Default value**

Inherit

### **Valid values**

Comma separated list of stylesheets.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **uiColor**

The toolbar area background color.

### **Default value**

Theme: Defaults to a color matching the current theme.

### **Valid values**

A color in #RRGGBB format.

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## 8.1.7 Clickable Image (std\_click\_image)

[QuickStart - Clickable Image Properties - Clickable Image](#)

The Clickable Image weblet provides a mechanism by which an image can be displayed on your web page that can be clicked on to perform an action. By default, it shows a blue ball that looks like this:



## QuickStart - Clickable Image

To use a clickable image with a column in a list, you would need to create a webroutine that specifies the list in its WEB\_MAP as \*BOTH or \*OUTPUT. When you open the generated XSL in the LANSAs Editor, you can change a column of the list to function as a clickable image as follows:

Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Clickable Image weblet.

Drag and drop it onto the column in your list you want to be clickable. The column's appearance changes to show that it is now an image. Click on an item in the column and then click on the Details tab. You should see that the name and value properties for the anchor weblet have already been set according to the field upon which it was dropped.

Set the currentrowhfield and currentrownumval properties as described in the property descriptions.

Set the on\_click\_wrname property to the name of the webroutine to be invoked when the hyperlink is clicked. If the webroutine is in a different WAM to the current webroutine then you will need to set the on\_click\_wamname property as well.

## Properties - Clickable Image

The Clickable Image weblet's properties are:

absolute_image_path	mouseover_absolute_image_path	relative_image_path
class	mouseover_relative_image_path	show_in_new_window
currentrowhfield	name	tab_index
currentrownumval	on_click_wamname	target_window_name
disabled	on_click_wrname	tooltip
disabled_class	pos_absolute	url
formname	presubmit_js	value
height_design	protocol	vf_wamevent
hide_focus	reentryfield	width_design
hide_if	reentryvalue	

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name in single quotes.

**value**

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field.

**Default value**

Blank.

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose one from a list).

**currentrowhfield**

The name of the field that will contain the current row's specified value. This property should only be used if the weblet is being used in a list.

**Default value**

'STDROWNUM' – in conjunction with the default value of position() for the currentrownumval property, STDROWNUM will hold the list entry number.

**Valid values**

The name of a repository- or WAM-defined field, in single quotes. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**currentrownumval**

The value to be placed in the field specified in the currentrowhfield property. This property should only be used if the weblet is being used in a list.

**Default value**

Position() – in conjunction with the default value of STDROWNUM for the currentrowhfield property, STDROWNUM will hold the list entry number.

**Valid values**

Any appropriate valid value. If specifying a field value from the current row of a list, prefix the name of the field with '\$'.

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

'STDREENTRY'

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

**tooltip**

Text to be displayed as a Tool Tip for the weblet when the mouse is hovered over it.

**Default value**

'Tooltip'

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **url**

This property may be used to specify a URL that the weblet will navigate to when clicked. If specified, the URL may be specified as a literal value (for example 'http://www.mycompany.com/') or a field name may be specified that contains the URL at run-time.

This property takes precedence over the `on_click_wamname`, `on_click_wrname` and `protocol` properties. The latter properties are ignored if `url` is specified.

### **Default Value**

'javascript:void();' – equivalent to nothing.

### **Valid Values**

A URL enclosed by single quotes or the name of a field, system variable or multilingual variable that will contain the URL at run-time.

**on\_click\_wamname**

The name of the WAM to be invoked when the weblet is clicked.

**Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_click\_wrname**

The name of the Webroutine to be invoked when the weblet is clicked.

**Default value**

Not applicable – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the on\_click\_wname property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **show\_in\_new\_window**

A Boolean property, the result of which determines whether response HTML for the weblet should be shown in a new browser window.

### **Default value**

false() – response HTML is shown in the current browser window.

### **Valid values**

true(), false() or any valid expression that returns True or False.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or any valid expression that returns True or False.

## **hide\_focus**

A Boolean property that, if evaluated to be True, will hide the focus rectangle for the weblet when it has focus.

### **Default value**

true()

### **Valid values**

true(), false() or a valid expression that returns a Boolean value.

## **relative\_image\_path**

The path and file name, relative to the 'images' directory, of the image to be displayed. If specified, the absolute\_image\_path property should be left blank.

### **Default value**

'ball\_blue.gif'

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **absolute\_image\_path**

The path and file name of the image to be displayed. If specified, the `relative_image_path` property should be left blank.

### **Default value**

The default is to use the image specified in the `relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

## **mouseover\_relative\_image\_path**

The path and file name, relative to the 'images' directory, of an image to be displayed when the mouse moves over the weblet.

### **Default value**

Blank – the image does not change when the mouse is moved over the weblet.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **mouseover\_absolute\_image\_path**

The path and file name of an image to be displayed when the mouse moves over the weblet.

### **Default value**

Blank – the default is to use the image specified in the `mouseover_relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_click\_image' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **disabled\_class**

The Cascading Style Sheet of the weblet when the disabled property is set to True.

### **Default value**

'std\_click\_image\_disabled' - The name of the shipped disabled class for the weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## presubmit\_js

JavaScript code to be run prior to the submission of the form.

### Default value

Blank. No JavaScript is run.

### Valid values

Any valid JavaScript function, or JavaScript code followed by a semicolon (;).

If you want to execute the presubmit JavaScript only, without running the JavaScript that submits the request (thus canceling the onclick event), append **return false;** to your presubmit JavaScript.

### Example

The following example shows a message box:

 presubmit_js	'alert("Hello world!");'
--	--------------------------

The following example shows a message box and cancels the submit JavaScript:

 presubmit_js	'alert("Hello world!"); return false;'
--	--

## **tab\_index**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

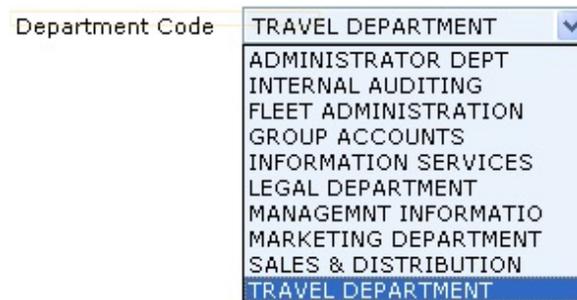
**Valid values**

String value. Comma (',') not allowed.

## 8.1.8 Combo Box (std\_dropdown)

### QuickStart - Clickable Image Properties - Combo Box

The combo box weblet builds a dropdown selection for a field. The values used to build the dropdown can be from a working list or a static set of values defined via the item property of the weblet. Each dropdown is implemented as a <select> HTML tag. It looks like this:



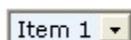
## QuickStart - Combo Box

Each entry in a combo box is defined by an entry in a working list or a set of items hardcoded in the combo box properties.

### If you use a working list:

To use a working list to define the dropdown options, you need to create a webroutine that specifies a field to store the selected value and the working list of options in the WEB\_MAP. When you open the XSL generated for the webroutine in the LANSA Editor:

1. If the working list was not \*HIDDEN on the WEB\_MAP a default table representation of the working list will be included on the web page. Delete the table that visualizes the list. To do this, right-click in the list and select Delete Entire List from the pop-up menu.
2. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Combo Box weblet.
3. Drag the Combo Box weblet onto the field to store the value and release the left-mouse button. This will display with dropdown options.



4. Click on the weblet to review the Details tab. Notice that the name and value properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected drop down entry. When the drop down value is changed the appropriate value will be placed in the field nominated on the name property – in this case the same field.
5. Change the listname property to the working list passed on the WEB\_MAP. The combo box representation should immediately change to represent the working list.
6. Set the codefield and captionfield properties to the appropriate fields from the working list.

### If you use the items property:

To use a set of items hardcoded in the combo box properties, you would need to create a webroutine that specifies a field in its WEB\_MAP. When you open the XSL generated for the webroutine in the LANSA Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list

near the top and locate the Combo Box weblet.

2. Drag the Combo Box weblet onto the field to store the value and release the left-mouse button. This will display with dropdown options.



3. Click on the weblet to review the Details tab. Notice that the name and value properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected drop down entry. When the drop down value is changed the appropriate value will be place in the field nominated on the name property – in this case the same field.
4. Set up the list of items to be used as drop down options by selecting the ellipses button on the items property. Proceed to define the require entries for the drop down.

## Properties - Combo Box

The Combo Box weblet's properties are:

captionfield	mouseover_class	selector_value_eq
class	name	submit_tagfields
codefield	on_change_wamname	tab_index
disabled	on_change_wrname	tagfield1
display_mode	pos_absolute	tagfield2
formname	protocol	tagfield3
hide_if	reentryfield	target_window_name
items	reentryvalue	value
listname	selector_field	vf_wamevent
		width_design

## **name**

The name of the dropdown. Normally, you would leave this as the default and let LANSA use its own internal naming convention. If the weblet has been dropped onto a field, or is to be used to display or populate a field, the field name is used.

## **Default value**

An automatically generated, unique identifier.

## **Valid values**

Single-quoted text.

## **Example**

This shows the default name is not associated with a field:

 name	concat('o', position(), '_LANSA_13186')
--	---

Or you when the weblet is associated with a field STD\_FLAG:

 name	'STD_FLAG'
--	------------

## value

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field or a default value.

## Default value

Blank.

## Valid values

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## Example

This example indicates the value should be set to the current value of the field #SECTION. When entered into the property this looks like this:

A screenshot of a property field. On the left, there is a small icon of a list with a green plus sign, followed by the text 'value'. To the right of this is a blue rectangular highlight. Further right, there is a text input field containing the text '#SECTION'.

When focus is moved off the property the same value will appear as follows:

A screenshot of a property field. On the left, there is a small icon of a list with a green plus sign, followed by the text 'value'. To the right of this is a white rectangular highlight. Further right, there is a text input field containing the text 'key('field-value', 'SECTION')'.

**display\_mode**

Controls whether the weblet accepts input or displays output.

**Default value**

'input'

**Valid values**

'input' or 'output'.

## items

An XML nodeset specifying the items to appear in the weblet. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. Leave blank if items are populated from a list specified in the listname property.

## Default value

document(")\*/lxml:data/lxml:dropdown (this indicates no items have been defined for this dropdown.)

## Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

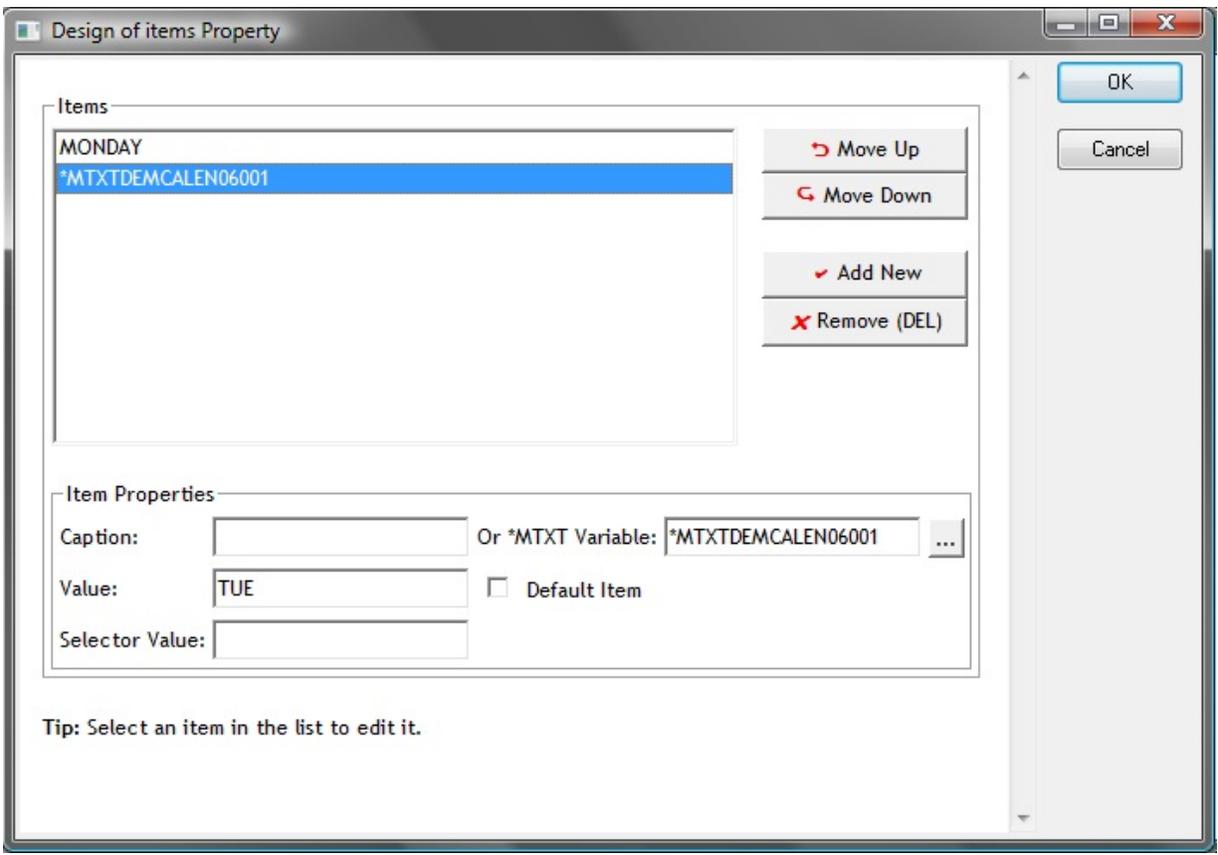
## Example

This example indicates that items have been setup in the designer to use as dropdown values.



The image shows a snippet of a property sheet. On the left, there is a small icon of a document with a green checkmark, followed by the text 'items'. To the right of this, the value of the property is displayed as 'document(")\*/lxml:data/lxml:caption\_value\_pairs[@id='4A14173FA6D64A059507F9124A630CF3']'. The entire snippet is enclosed in a light gray border.

Using the ellipse button on the property you will see the designer and be able to maintain the items to be displayed in the dropdown. The following view of the designer indicates two entries have been set up for the dropdown. The first entry has the literal value 'MONDAY' and the second entry uses a multilingual variable to display the description for the code TUE. Check the Default Item check box for the item which is to be selected if no value is preselected.



**listname**

The name of the working list to use to populate the cells in the grid. Leave blank if details are specified in the items property.

**Default value**

Blank.

**Valid values**

Single-quoted text. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

**selector\_field**

The name of the field in the list specified in the listname property that can contain a value to limit, to a subset, the list items shown in the weblet. This property is used in conjunction with the selector\_value\_eq property.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **selector\_value\_eq**

This value is used in order to limit, to a subset, the list items shown in the weblet. If a listname property is provided the associated field must be specified in the selector\_field property. If the items property designer has been used to define the list of values the corresponding selector value entered in the designer is used.

### **Default value**

Blank.

### **Valid values**

Single-quoted text or a numeric value. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**codefield**

The name of the field in the list specified in the listname property that holds the key value for each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**captionfield**

The name of the field in the list specified in the listname property that holds the caption for the each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **tagfield1**

The name of a field in the list specified in the listname property that can contain an additional value to be tagged onto a list item. This value is added as an attribute with the name of the field prefixed with 'tag\_'. This attribute can then be specified in JavaScript code.

### **Default value**

Blank.

### **Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **tagfield2**

The name of a field in the list specified in the listname property that can contain an additional value to be tagged onto a list item. This value is added as an attribute with the name of the field prefixed with 'tag\_'. This attribute can then be specified in JavaScript code.

### **Default value**

Blank.

### **Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **tagfield3**

The name of a field in the list specified in the listname property that can contain an additional value to be tagged onto a list item. This value is added as an attribute with the name of the field prefixed with 'tag\_'. This attribute can then be specified in JavaScript code.

### **Default value**

Blank.

### **Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **submit\_tagfields**

A Boolean property. Set to True if tag field values are to be submitted with the rest of the form.

### **Default value**

True().

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

'STDREENTRY'

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## hide\_if

A Boolean property. An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the grid will always be shown)

### Valid values

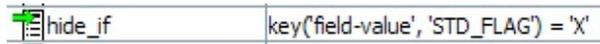
True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the grid if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:



When the property loses focus, the expression is shown as follows:



**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognised. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

## **on\_change\_wamname**

The name of the WAM to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a dropdown. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_change\_wrname**

The name of the Webroutine to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a dropdown. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

Blank.

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_change\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown. A unique name can be entered or use the available selection for a predefined set of values.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes.

A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

**class**

The Cascading Style Sheet class to be applied to the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**mouseover\_class**

The Cascading Style Sheet class to be applied to the weblet when the mouse is moved over it.

**Default value**

Blank.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_dropdown\_mouseover' is supplied.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.9 Dynamic Select Box (std\_dynamic\_select)

[QuickStart - Dynamic Select Box](#)

[8.1.9 Dynamic Select Box \(std\\_dynamic\\_select\)](#)

The Dynamic Select Box is an HTML <select> element (which means it can create a dropdown or a list) that is able to monitor another field and automatically refresh itself when that field changes. The values used to build the list can be from a working list or a static set of values defined via the item property of the weblet.

## QuickStart - Dynamic Select Box

Each entry in a Dynamic Select Box is defined by an entry in a working list or a set of items hardcoded in the weblet properties.

### If you use a working list:

To use a working list to define the dropdown options, you need to create a webroutine that specifies a field to store the selected value and the working list of options in the WEB\_MAP. The working list must be defined as a \*JSON list. The working list will usually contain 2 or 3 columns:

A caption column containing the values to display in the list

A code column containing the code associated with each caption. The code is the value that will be sent back to indicate the user's choice.

An optional selector column. This will be used in conjunction with the selectorValueField property to filter the list displayed to the user

1. If the working list was not \*HIDDEN on the WEB\_MAP a default table representation of the working list will be included on the web page. Delete the table that visualizes the list. To do this, right-click in the list and select Delete Entire List from the pop-up menu.
2. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Dynamic Select Box weblet.
3. Drag the Dynamic Select Box weblet onto the field to store the value and release the left-mouse button.
4. Click on the weblet to review the Details tab. Notice that the [vf\\_wamevent](#) and [value](#) properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected entry. When the weblet value is changed the appropriate value will be placed in the field nominated on the name property – in this case the same field.
5. Change the listname property to the working list passed on the WEB\_MAP.
6. Set the codefield and captionfield properties to the appropriate fields from the working list.
7. Set the size property to indicate the desired height of the list box in rows (a value of 1 will cause it to render as a dropdown).

## If you use the items property:

To use a set of items hardcoded in the weblet properties, you would need to create a webroutine that specifies a field for the selected value in its WEB\_MAP. When you open the XSL generated for the webroutine in the LANSAs Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Dynamic Select Box weblet.
2. Drag the Dynamic Select Box weblet onto the field to store the value and release the left-mouse button.
3. Click on the weblet to review the Details tab. Notice that the `vf_wamevent` and `value` properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected entry. When the weblet value is changed the appropriate value will be placed in the field nominated in the name property – in this case the same field.
4. Set up the list of items to be used as drop down options by selecting the ellipses button on the items property. Proceed to define the required entries for the select box.

## Automatic Updating

The dynamic select box can monitor another field and automatically refresh itself whenever that field is updated. If the weblet has been filled using a working list then you will need to create a JSON webroutine that will output a fresh copy of the working list. The weblet will call this Webroutine each time it needs to refresh.

1. Set the `updateOnFieldChange` to the id of the field you want to monitor. This will usually be the name of a field but it can be the id of any HTML form element capable of generating a change event. This field will be submitted to the Webroutine supplying the new list.
  2. Set the `updateWamName` and `updateWrName` properties to the name of the WAM/Webroutine that will supply the new list. This webroutine must output a new copy of the list defined in the listname property.
  3. Set the `updateFieldsToSubmit` property to identify the input values to be sent to the update webroutine.

It is not necessary to use an update webroutine to refresh the list. In many simple cases you can supply all possible values to the weblet in the initial list

output by your main webroutine and filter the list using the [selectorField](#). The weblet will remember the initial list and reapply the selectorField filter when doing a refresh, avoiding the need to send a request to the server and wait for a response.

For example, if the [updateOnFieldChange](#) and [selectorValueField](#) properties both specify the same field, the list will update itself by re-filtering every time the user changes that field.

Deciding which approach to take is a balancing act. Using the selectorField makes the list update more quickly but may result in a longer initial page load as more data has to be sent to the browser. Using an update webroutine will improve initial load and allows real time retrieval of the latest data or allows more complex logic in the list construction but may introduce a delay while the list is retrieved from the server.

**Also see**

[QuickStart - Dynamic Select Box](#)

## Properties - Dynamic Select Box

The Dynamic Select Box weblet's properties are:

allowMultiSelect	multiSelectListname	size
captionField	name	tabIndex
class	onChangeExtraFields	updateFieldsToSubmit
codeField	onChangeFormname	updateOnFieldChange
disabled	onChangeProtocol	updateProtocol
display_mode	onChangeTarget	updateWamName
hide_if	onChangeWamName	updateWrName
id	onChangeWrName	value
items	position	vf_wamevent
listname	selectorField	width
multiSelectCodefield	selectorValueField	

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. If the weblet has been dropped onto a field, or is to be used to display or populate a field, the field name is used.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Any string starting with a letter ([A-Za-z]) followed by any number of letters, digits([0-9]), hyphens ("-") or underscores ("\_").

**id**

A unique id for the weblet. The default is the same as the name property and normally you would leave it as that. In some special circumstances you may have multiple weblets, in multiple forms, visualizing the same field. In those cases you would need to set this property to give each one a unique ID.

**Default value**

\$name The same as the name property

**Valid values**

Any string starting with a letter ([A-Za-z]) followed by any number of letters, digits([0-9]), hyphens ("-") or underscores ("\_").

**value**

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field or a default value.

**Default value**

Blank.

**Valid values**

Any text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

**size**

The height (expressed as a number of lines) of the weblet. If the value is 1, the weblet will display as a drop-down list. If it is greater than 1, it will display as a list box.

**Default Value**

The weblet displays as a drop-down list.

**Valid values**

An integer value greater than 0.

**display\_mode**

Controls whether the weblet accepts input or displays output.

**Default value**

'input'

**Valid values**

'input' or 'output'.

**hide\_if**

A Boolean property. An expression which, if evaluated to be True, will hide the weblet.

**Default value**

false() (that is, the weblet will always be shown)

**Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

## items

An XML nodeset specifying the items to appear in the weblet. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. Leave blank if items are populated from a list specified in the listname property.

## Default value

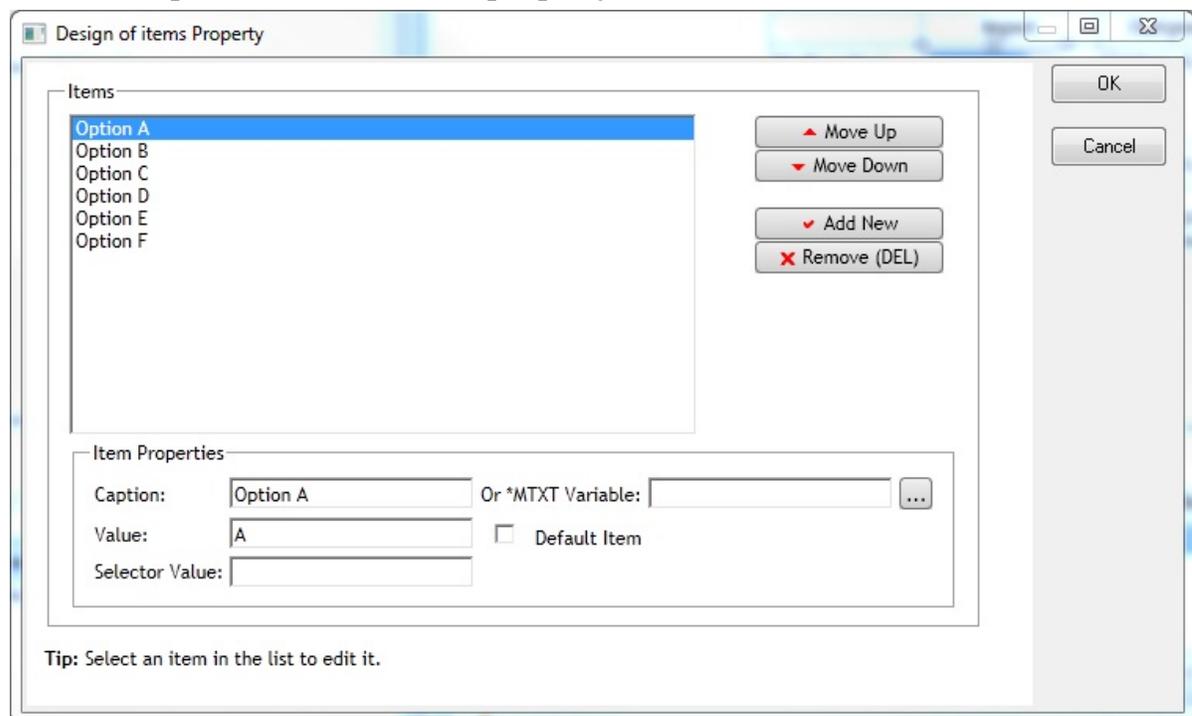
document(")/\*/lxml:data/lxml:select (this indicates no items have been defined for this weblet.)

## Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## Example

This example shows the item's property editor:



This shows a list configured with 6 items. Check the Default Item check box for the item which is to be selected if no value is preselected. The Selector value can be used to filter the list down to a smaller set of displayed values at runtime.

**listname**

The name of the working list to use to populate the weblet list. Leave blank if details are specified in the items property. If both the listname and items properties are specified, the listname property will take priority.

**Default value**

Blank.

**Valid values**

Blank, or the name of an output working list that is defined as \*JSON in the current webroutine.

## **selectorField**

The name of the field in the [listname] working list that contains a selector value. The selector value is used to filter the working list into a smaller list of values that are actually displayed at runtime. If the value of the selectorField column matches the value in selectorValueField then the entry will be included in the list.

## **Default value**

Blank.

## **Valid values**

A field, from the [listname] working list. A value can be selected by clicking the corresponding dropdown button in the property sheet.

## **selectorValueField**

The name of a field whose value is used to filter the list supplied to the weblet into a smaller list for display. When building the display list, this value is compared with the value in the lists "selector" column. If a match is found the entry is included in the displayed list.

This can be useful for reducing the work done at the server. Instead of calculating the list entries every time it is executed, the webroutine could output a pre-built list with all possible values and a selector value. The browser can then reduce the list to a subset based on the selector value.

This can also be used to allow a dynamic list to refresh without having to make a server request. If the field being monitored for updates is also the selectorValueField then the weblet can rebuild itself by applying the new selector value to the list initially passed to it.

### **Default value**

Blank. No filtering is done.

### **Valid values**

The name of any output field in the current Webroutine.

**codeField**

The name of the field in the [listname] working list that holds the key value for each list item.

**Default value**

Blank.

**Valid values**

A field, from the [listname] working. A value can be selected by clicking the corresponding dropdown button in the property sheet.

**captionField**

The name of the field in the [listname] working list that holds the caption value for each list item.

**Default value**

Blank.

**Valid values**

A field, from the [listname] working. A value can be selected by clicking the corresponding dropdown button in the property sheet.

## **allowMultiSelect**

A Boolean property that controls whether multiple selections are allowed in the list box. If multiple selections are allowed, the multiSelectListname and multiSelectCodefield properties must be specified. Note that only a listbox is capable of multiple selections. If size is 1, this property will be ignored.

### **Default value**

false() – only a single selection may be made in the list box.

### **Valid values**

true(), false(), or a valid expression that returns True or False.

## **multiSelectListname**

The working list that contains the selected entries for the list box. The working list should contain only the code field that is specified in the multiSelectCodefield property. If allowMultiSelect is false or size is 1, this property is ignored.

### **Default value**

Blank – only a single selection may be made in the list box.

### **Valid values**

The name of a working list. Click the corresponding dropdown button in the property sheet to choose from a list of known working lists.

## **multiSelectCodefield**

The name of the field in the [multiSelectListname] working list that holds the code value of the selected list box items.

### **Default value**

Blank – only a single selection may be made in the list box.

### **Valid values**

The name of a field. Click the corresponding dropdown button in the property sheet to choose from a list of known fields.

## **onChangeWamName**

The name of the WAM to be invoked when an item in the weblet is selected.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **onChangeWrName**

The name of the Webroutine to be invoked when an item in the weblet is selected.

### **Default value**

Blank.

### **Valid values**

The name of a Webroutine. The Webroutine must exist in the WAM specified in the onChangeWamName property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **onChangeFormname**

The name of the HTML form to post to the server when calling the onChange webroutine. Normally you will not need to change this property. Advanced applications with multiple forms may need it to ensure the correct form is sent.

### **Default value**

'LANSA' (that is, document.LANSA)

### **Valid values**

A name for the form. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## onChangeExtraFields

An XML nodeset specifying any extra fields (not already in the form being submitted) that should be sent to the onChange webroutine. This will most commonly be used when the weblet is used in a list or grid to specify values from other columns in the list.

### Default value

document(")/\*/lxml:data/lxml:json[not(@id)] (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

### Example

This example shows the onChangeExtraFields property editor:

```
Webroutine Name(CheckConsignment)
  Web_Map For(*input) Fields(#WDCONSIGN)

  * see if this consignment exists in the consignment status file.
  Check_For In_File(WDCONST) With_Key(#WDCONSIGN)

  * it exists ...
  If_Status Is(*EQUALKEY)
    * show the consignment status.
    Transfer Toroutine(ShowConsignment)
  Else
    * it doesn't exist: show an error message ...
    Message Msgtxt('Invalid Consignment Note number. Please try again.')

    * and re-display the main page.
    Transfer Toroutine(ConsignmentEnquiry)
  Endif
Endroutine
```

This shows how output fields in the current webroutine (the "Value" column) can be mapped to input fields with a different name (the "Name" column) defined in the onChange webroutine's WEB\_MAP.

## **onChangeProtocol**

The protocol (for example, http:// or https://) that should be used when calling the onChange webroutine.

### **Default value**

Blank. This is equivalent to the current protocol being used.

### **Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **onChangeTarget**

The name of the window, or frame, in which response HTML will be shown. A unique name can be entered or use the available selection for a predefined set of values.

### **Default value**

Blank – response HTML will be shown in the current window/frame.

### **Valid values**

The name of a window or frame, in single quotes.

A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

**position**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognised. The property will usually be set in pixels by dragging and dropping the weblet.

**Default value**

Blank (not positioned).

**Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement.

**width**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement.

**disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

**Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

**Valid values**

true(), false() or a valid expression.

**class**

A CSS class to be applied to the weblet.

**Default value**

Blank

**Valid values**

Any valid class name from the current Cascading Style Sheet. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tabIndex**

Determines the tab order of the weblet on the form. The tabIndex property value determines the tab order as follows:

1. Objects with a positive tabIndex are selected in increasing tabIndex order (and in source order to resolve duplicates).
2. Objects with a tabIndex of zero or blank (the default) are selected in source order.
3. Objects with a negative tabIndex are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **updateOnFieldChange**

The ID of a field to monitor for changes. If a change occurs in the monitored field the select box will refresh.

### **Default value**

blank

### **Valid values**

The ID of any field on the current page capable of generating "change" events. That means any text input fields, or select elements (dropdown lists or list boxes). If updateWamName and updateWrName have been specified, the weblet will call the webroutine to request a fresh copy of the [listname] working list. Otherwise it will re-apply the selectorValue filter to the list it already has and rebuild the select list from that.

## **updateWamName**

The name of the WAM to be invoked when refreshing the list.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **updateWrName**

The name of the Webroutine to be invoked when refreshing the list. This webroutine must be defined as \*JSON.

### **Default value**

Blank.

### **Valid values**

The name of a Webroutine. The Webroutine must exist in the WAM specified in the updateWamName property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## updateFieldsToSubmit

An XML nodeset specifying any fields that should be sent to the update webroutine.

### Default value

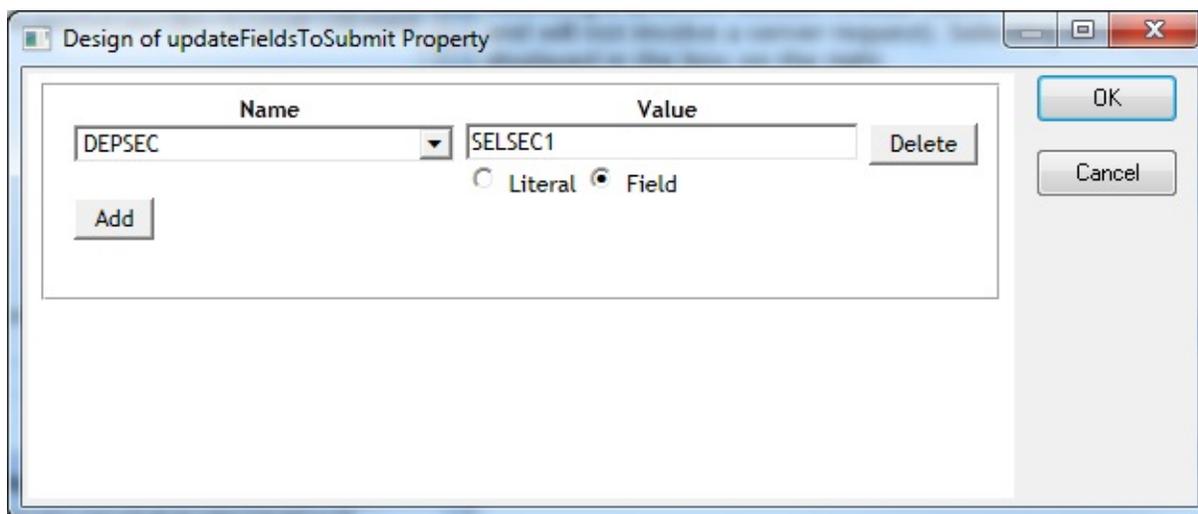
document(")/\*//lxml:data/lxml:json (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

### Example

This example shows the updateFieldsToSubmit property editor:



This shows how output fields in the current webroutine (the "Value" column) can be mapped to input fields with a different name (the "Name" column) defined in the update webroutine's WEB\_MAP.

## **updateProtocol**

The protocol (for example, http:// or https://) that should be used when calling the update webroutine.

### **Default value**

Blank. This is equivalent to the current protocol being used.

### **Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.10 Export to Excel (std\_toexcel)

[QuickStart](#) – [Export to Excel](#) [Properties](#) – [Export to Excel](#)

The Export to Excel weblet allows you to export a table or grid to an Excel spreadsheet.

A rectangular button with rounded corners, a light blue gradient background, and a thin dark border. The text "Export to Excel" is centered on the button in a dark blue, sans-serif font.

Export to Excel

## QuickStart – Export to Excel

You would typically use this weblet when your page has a table or grid with tabular data that users would want to manipulate in a spreadsheet.

This weblet uses ActiveX and works only in MS Internet Explorer. The weblet is disabled if the browser doesn't support ActiveX.

This weblet is to be used with tables that contain output fields only. If the table contains Weblets or input elements, the result won't work as expected.

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Export to Excel weblet.
2. Drag and drop the weblet onto the web page.
3. Set the listname property with the name of the list you want to make exportable.
4. Set the startingColumnIndex property with the index of the first column to include in the export (first column has index 0).
5. Set the numberOfColumns property with the numbers of columns to export.
6. Change the caption property if needed.

## Properties – Export to Excel

The Export to Excel properties are:

caption	listname	startingColumnIndex
disabled	name	tab_index
height_design	numberOfColumns	text_class
hide_if	pos_absolute	title
		width_design

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name enclosed in single quotes.

**listname**

The name of the working list to export. This property is required.

**Default value**

Blank.

**Valid values**

Single-quoted text. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

**startingColumnIndex**

The index of the column from which to start the export. The index of the first column is zero.

**Default value**

0.

**Valid values**

An integer value. Must be less than the number of columns in the table.

## **numberOfColumns**

The number of columns to include in the export

### **Default value**

last – All columns from the starting column specified in startingColumnIndex up to the last column in the table.

### **Valid values**

An integer value. It should not exceed the last column, starting from the column specified in startingColumnIndex.

**caption**

The caption for the weblet.

**Default value**

'Export to Excel'

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled. If the browser doesn't support ActiveX, the weblet is automatically disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or any valid expression that returns True or False.

**title**

Text to be displayed as a Tool Tip for the weblet when the mouse is hovered over it.

**Default value**

Blank – no Tool Tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**text\_class**

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

Blank – no text class

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

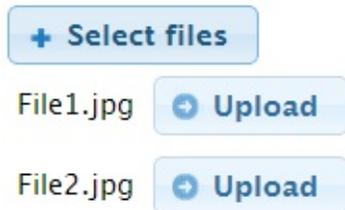
### **Valid values**

Blank or a valid numeric value.

## 8.1.11 File Upload (std\_fileupload)

### QuickStart – File Upload Properties – File Upload

The file upload weblet allows you to select files to upload to the application server (into a temporary directory). The webroutine that receives the file upload can then manipulate the uploaded files as required.



## **QuickStart – File Upload**

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the File Upload weblet.
2. Drag and drop the weblet onto your page in the Design view. Make sure the weblet is selected and then click on the Details tab.
3. Set the name of the weblet. Assign it the name of the LOB field that will receive the file path in your webroutine (see uploadWrName property)
4. Set the WAM and webroutine name to receive the file upload.
5. On the designer, make sure the weblet is selected and right click to select "Create Ajax Webroutine". This will generate a skeleton for the webroutine handling the file upload.

## Properties – File Upload

caption	MaxFileSize	tab_index
class	MaxNumberOfFiles	text_class
failCallback	name	uploadWamName
hide_If	successCallback	uploadWrName
id		

**name**

The name of LOB field to receive the uploaded file temporary path.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **id**

A unique id for the weblet. The default is the same as the name property and normally you would leave it as that. In some special circumstances you may have multiple weblets, in multiple forms, visualizing the same field. In those cases you would need to set this property to give each one a unique ID.

### **Default value**

\$name The same as the name property

### **Valid values**

Any string starting with a letter ([A-Za-z]) followed by any number of letters, digits([0-9]), hyphens ("-") or underscores ("\_").

**caption**

Specifies the caption for the button to add files.

**Default value**

Default: "Select files"

**Valid values**

Any string value.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**hide\_If**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **uploadWamName**

The name of the WAM whose Webroutine receives the file uploaded by this weblet.

### **Default value**

The current WAM

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **uploadWrName**

The name of the Webroutine that receives the file uploaded by this weblet. It must be a JSON webroutine. Its response is passed to the optional JavaScript callback functions to provide feedback to the user.

### **Default value**

Blank – a Webroutine name must be specified.

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must be a JSON response weboutine and exist in the WAM specified in the uploadWamName property. A list of known JSON Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **MaxFileSize**

The maximum file size allowed in Megabytes.

### **Default value**

5 – 5Mb

### **Valid values**

An integer value. Must be consistent with maximum value for file uploads defined in the application server. Note: Browsers may have their own limits.

## **MaxNumberOfFiles**

The maximum number of files allowed.

### **Default value**

1

### **Valid values**

An integer value.

## **successCallback**

The name of the optional JavaScript function to call when the file is successfully uploaded. The function is called with two arguments: The event object (null if not available) and the JSON webroutine response from the file upload webroutine.

## **Default Value**

None

## **Valid values**

The name of a JavaScript function.

## **failCallback**

The name of the optional JavaScript function to call when the file upload fails. The function is called with two arguments: The event object (null if not available) and a constructed JSON webroutine response with the error messages (mimics messages issued by a webroutine).

## **Default Value**

None

## **Valid values**

The name of a JavaScript function.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled. If the browser doesn't support ActiveX, the weblet is automatically disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or any valid expression that returns True or False.

**text\_class**

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

Blank – no text class

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## 8.1.12 Grid (std\_grid\_v2 and std\_grid\_v3)

### QuickStart - Grid Properties - Grid

The grid weblet provides a grid control with sortable columns. Grid cells are populated from a working list or XML. Grid elements are implemented as a HTML <table>. Use std\_grid\_v3 weblet if you want to use jQuery UI based weblets. It looks like this:

Delete?	Skill Code	Skill Description
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>		
<input type="checkbox"/>	4GL	4GL PROGRAMMING
<input type="checkbox"/>	AC	ACCOUNTANCY DEGREE
<input type="checkbox"/>	ADMIN1	ADMINISTRATN PART 1
<input type="checkbox"/>	ADMIN2	ADMINISTRATN PART 2
<input type="checkbox"/>	ADVPGM	ADVANCED PROGRAMMING
<input type="checkbox"/>	CL	CL PROGRAMMING
<input type="checkbox"/>	COM	COMMUNICATIONS DEGREE
<input type="checkbox"/>	CS	COMPUTER SCIENCE DEGREE

## QuickStart - Grid

To use this weblet you would typically create a webroutine that specifies a working list in its WEB\_MAP. The grid weblet can be used to represent the data in the working list by completing the following steps in the LANSAs Editor:

1. If the working list was not \*HIDDEN on the WEB\_MAP a default table representation of the working list will be included on the web page. Delete the table that visualizes the list. To do this, right-click in the list and select Delete Entire List from the pop-up menu.
2. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Grid weblet.
3. Drag and drop the weblet onto your page in the Design view. Make sure the weblet is selected and then click on the Details tab.
4. Set the listname property to the working list passed on the WEB\_MAP. The grid representation should immediately reflect the working list details.
5. Optionally you may want to set the height and width properties to fit your webpage design.

For more information on customizing grid columns with your own weblets refer to [Customize Grid Columns](#).

## Properties - Grid

The Grid weblet's properties are:

allowColResize	hide_header_if_empty	pos_absolute
allowSort	hide_if	rowHoverEffect
even_row_class	listname	selectableRow
formname	listname_fixed_col_field	show_header
grid_col_properties	name	sort_fixed_rows_with_body
grid_hdr_properties	odd_row_class	width
height	onRowClickJS	

**name**

The name of the grid. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the grid.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**listname**

The name of the working list to use to populate the cells in the grid. If specified, the `or_list_xml` property should be left blank.

**Default value**

Blank.

**Valid values**

Single-quoted text. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

## **listname\_fixed\_col\_field**

Specify the field name, associated with the list specified in listname, that will contain the entries to appear in the leftmost fixed (non-scrollable) column.

### **Default value**

Blank.

### **Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **sort\_fixed\_rows\_with\_body**

A Boolean property. If false() the entries in the fixed column specified in listname\_fixed\_col\_field property, will not move with the rest of the row when the other columns in the grid are sorted.

### **Default value**

True()

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

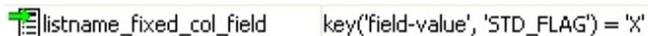
### **Example**

This example will sort the fixed column with the other columns if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. The left side shows a list of properties with 'listname\_fixed\_col\_field' selected. The right side shows the expression '#STD\_FLAG = 'X''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. The left side shows a list of properties with 'listname\_fixed\_col\_field' selected. The right side shows the expression 'key('field-value', 'STD\_FLAG') = 'X''.

## grid\_hdr\_properties

Used to set the properties of grid header columns. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet.

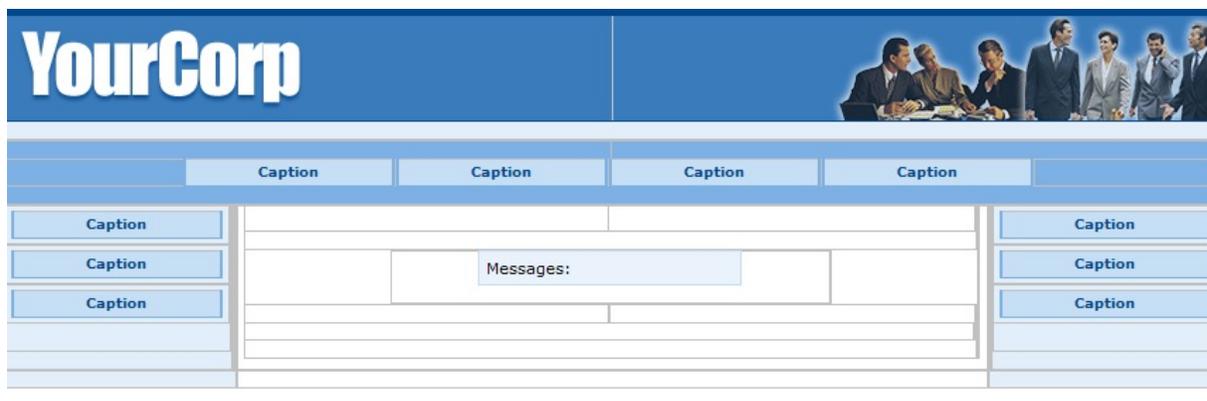
### Default value

document(")/\*/lxml:data/lxml:grid[@id='<unique id>'] - indicates the lxml fragment relating to this definition is store in the current document under the lxml data element and is identified as a grid with a unique id as generated.

### Valid values

Not Applicable. (This value is system maintained.) Use the ellipses button in the property sheet to select grid columns and indicate the properties to be maintained.

In this example the field SURNAME will allow the column headings to be customized and the column will be sorted in reverse order when clicked:



### Example

The following example indicates that an LXML fragment with a unique identifier matching the value indicated by @id had been automatically generated in the current document to define the grid header properties.

```
grid_hdr_properties document(")/*/lxml:data/lxml:grid[@id='D574DF799BD145998BE947F0F4D45944']
```

You can review these LXML fragments by opening the XSL Source tab and searching for references to the unique identifier.

## grid\_col\_properties

Used to set properties of the grid columns. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet.

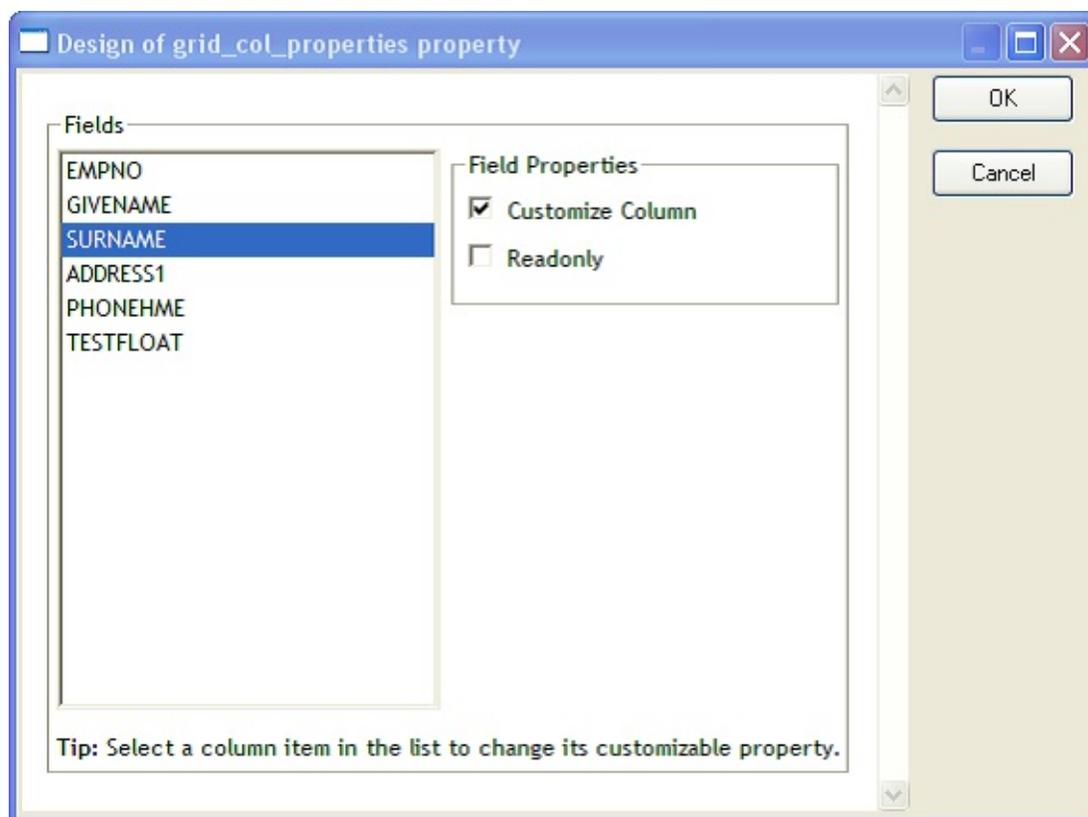
### Default value

document('')/\*/lxml:data/lxml:grid[@id='<unique id>'] - indicates the lxml fragment relating to this definition is store in the current document under the lxml data element and is identified as a grid with a unique id as generated.

### Valid values

Not Applicable. (This value is system maintained.) Use the ellipses button in the property sheet to select grid columns and indicate the properties to be maintained.

In this example the field SURNAME will allow the column to be customized. It is not read only.



### Example

The following example indicates that an LXML fragment with a unique identifier matching the value indicated by @id had been automatically

generated in the current document to define the grid column properties.

A screenshot of an XSL Source tab in a software application. The search bar contains the text 'grid\_col\_properties' and the search results show the XPath expression 'document('')/\*//xml:data/xml:grid[@id='7C4F340984F6475BBD90CA28416F3F22']'. The search results are displayed in a table with two columns: the search term and the corresponding XPath expression.

Search	Results
grid_col_properties	document('')/*//xml:data/xml:grid[@id='7C4F340984F6475BBD90CA28416F3F22']

You can review these LXML fragments by opening the XSL Source tab and searching for references to the unique identifier.

## show\_header

A Boolean property. If false(), the column headers will not be shown in the grid.

### Default value

true()

### Valid values

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will show the column headings if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

A screenshot of a property editor. On the left, there is a small icon and the text 'show\_header'. A blue highlight bar is positioned over this text. To the right of the highlight bar, the expression '#STD\_FLAG = 'X'' is displayed.

show\_header #STD\_FLAG = 'X'

When the property loses focus, the expression is shown as follows:

A screenshot of a property editor. On the left, there is a small icon and the text 'show\_header'. To the right of this text, the expression 'key('field-value', 'STD\_FLAG') = 'X'' is displayed.

show\_header key('field-value', 'STD\_FLAG') = 'X'

## hide\_header\_if\_empty

A Boolean property. If true(), the column headers will not be shown in the grid, if there are no entries in the list, specified in listname property.

### Default value

true()

### Valid values

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will show the column headings, even when there are no list entries to display, if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

hide\_header\_if\_empty #STD\_FLAG = 'X'

## hide\_if

A Boolean property. An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the grid will always be shown)

### Valid values

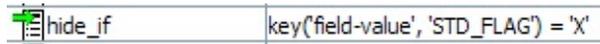
True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the grid if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:



When the property loses focus, the expression is shown as follows:



## **even\_row\_class**

The Cascading Style Sheet class to be applied to the even number grid rows.

### **Default value**

'even\_row'. This is the default class applied to even numbered rows (that is, 2<sup>nd</sup>, 4<sup>th</sup> row etc.) in the grid and is provided with the all shipped cascading styles sheets.

### **Valid values**

Any valid class name selected from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **odd\_row\_class**

The Cascading Style Sheet class to be applied to the odd number grid rows.

### **Default value**

'odd\_row'. This is the default class applied to odd numbered rows (that is, 1<sup>st</sup>, 3<sup>rd</sup> etc...) in the grid and is provided with the all shipped cascading styles sheets.

### **Valid values**

Any valid class name selected from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognised. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

Blank (this is equivalent to the weblet adopting its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the weblet on the web page.

**Default value**

Blank (this is equivalent to the weblet adopting its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## **allowSort**

A Boolean property. If true(), the user will be able to sort the grid contents by clicking on a column heading.

### **Default value**

true()

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

## **allowColResize**

A Boolean property. If true(), the user will be able to resize grid columns by clicking and dragging the border between header cells.

### **Default value**

true()

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

## **rowHoverEffect**

A Boolean property. If true(), the grid will provide some visual feedback to the user by highlighting the row under the mouse pointer.

### **Default value**

false()

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

## **selectableRow**

A Boolean property. If true(), the grid will track the row last clicked on and highlight that row to indicate selection.

### **Default value**

false()

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

## **onRowClickJS**

JavaScript code to be run when the user clicks on a row in the grid. The code receives two variables that provide extra information about the click event:

event – The browser Event object for the click event.

rowNum – the number of the row clicked on. This will be the number of the row in the working list, not the row within the grid (which may change with user sorting).

You can prevent the row being selected by returning false from this code.

### **Default value**

Blank

### **Valid values**

Any valid JavaScript code

### **Example**

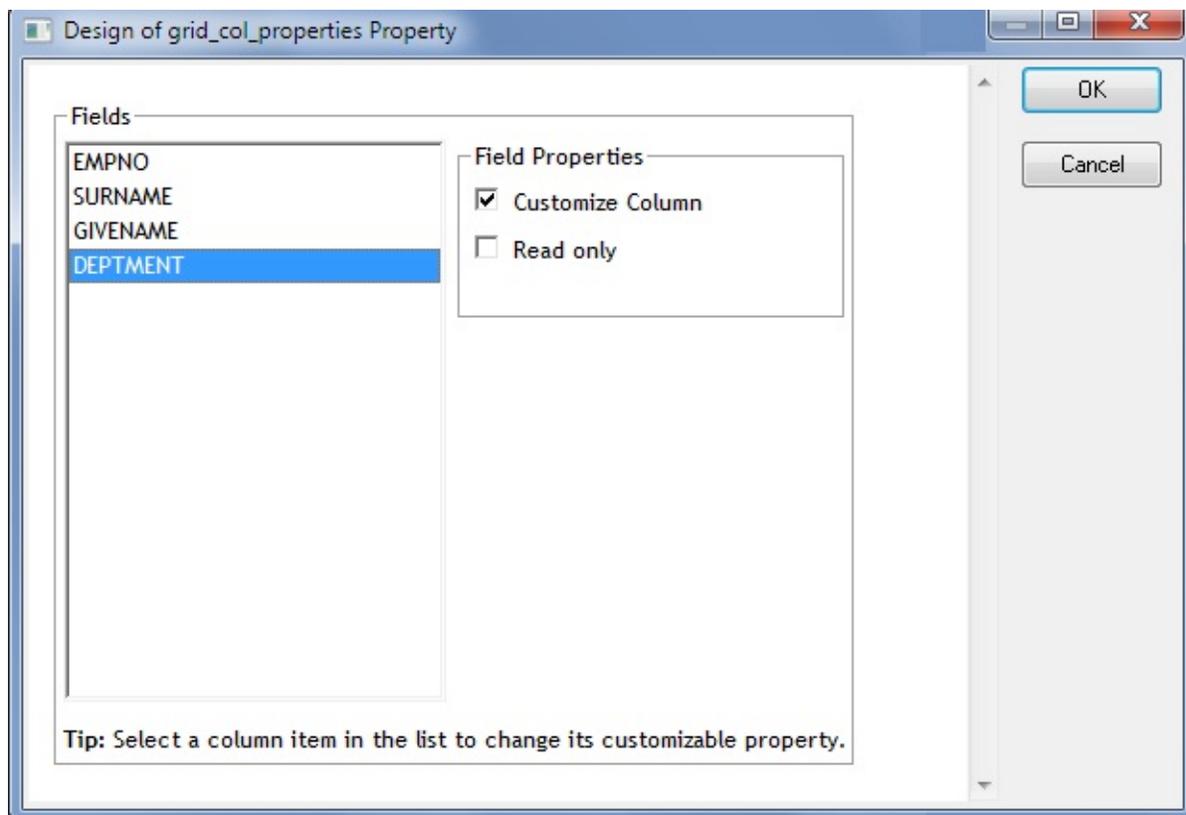
The following will display an alert indicating the row clicked, and prevent row 3 from being selected:

```
alert("Row " + rowNum); return (rowNum != 3);
```

## Customize Grid Columns

### [Reference Column Values in Weblet Properties](#) [Grid Column Example](#)

The grid will automatically visualize each column using the default weblet for that field. Sometimes you may want to change the properties of the weblet or use a different weblet. To do this, edit the `grid_col_properties` property and select the `Customize Column` option for the column you wish to customize.



When you click OK, the customized columns will go blank.

# LANSA

Advanced Software Made Simple

Messages:

## Employee List

Employ Number	Surname	Given name(s)	Dept Code
ABCDE	ABCDEFGHIJKLMNOPS	ABCDEFGHIJKLMNOPS	
ABCDE	ABCDEFGHIJKLMNOPS	ABCDEFGHIJKLMNOPS	
ABCDE	ABCDEFGHIJKLMNOPS	ABCDEFGHIJKLMNOPS	

Now you can add any Weblet to the column by dragging it to any one of the empty cells in the column.

## Reference Column Values in Weblet Properties

Referencing column values in a grid is a little different than you may have seen before. When using a standard list you can reference the value of a column using an XSLT variable of the same name (\$COLUMNNAME). This is not possible with the grid so it is necessary to access a column with the following XPath expression:

```
../lxml:column[@name='COLUMNNAME']
```

This is not necessary for referencing the current column (i.e. the column containing the weblet). In this case you can use a single period (.).

If you know the position of the column you want to reference, you can use an XPath expression like this:

```
../lxml:column[2]
```

This is handy for large lists as it is much faster but it may cause problems if you change the order of the columns in your list.

**Note:** All field and column name references in XPath expressions must be uppercase. All references to repository fields must use the object name for the field.

### Also see

[confirmText](#)

The XPath expression tells the XSLT processor where to find the column in the XML data output by the Webroutine. Like DOS or Linux file paths, the expression indicates a path to the target data from the current position and, like file paths, a period (.) refers to the current position and two periods (..) refer to the parent. Following is the XML output for a single row in the list:

```
<lxml:entry>
  <lxml:column name="EMPNO"
id="EMPLIST.0002.EMPNO">A0090</lxml:column>
  <lxml:column name="SURNAME"
id="EMPLIST.0002.SURNAME">BLOGGS</lxml:column>
```

```
<lxml:column name="GIVENAME"
id="EMPLIST.0002.GIVENAME">FRED JOHN
ALAN</lxml:column>
  <lxml:column name="DEPTMENT"
id="EMPLIST.0002.DEPTMENT">FLT</lxml:column>
</lxml:entry>
```

When processing a weblet in a grid column, the current position in the XML is the `<lxml:column>` tag for that row and column. An XPath expression like this:

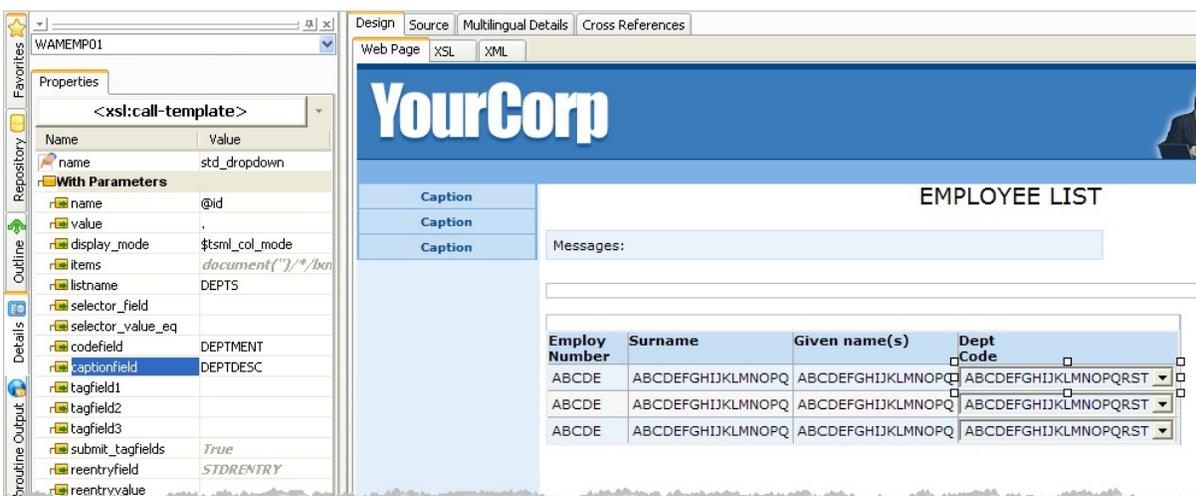
```
../lxml:column[@name='EMPNO']
```

says to go up to the parent `<lxml:entry>` tag, look for a child `<lxml:column>` tag with a name attribute of 'EMPNO' and return it.

## Grid Column Example

This example adds a combo box to the Department column in a list of employees. It assumes an employee list called #EMPLIST containing a #DEPARTMENT column and a department list called #DEPTS containing #DEPARTMENT and #DEPTDESC.

1. Add a grid to the Webroutine design and set its *listname* property to **EMPLIST**.
2. Modify the *grid\_col\_properties* property and select the Customize Column option for the *DEPARTMENT* column.
3. Drag a combo box weblet onto the first non-header cell of the *DEPARTMENT* column.
4. Set the *listname* property of the dropdown to **DEPTS**.
5. Set the *codefield* property to **DEPARTMENT** and the *captionfield* property to **DEPTDESC**. Your grid should be looking something like this:



Notice that the *value* property contains a single period (.). This indicates that the value for the combo box should be the value of the current column of the *EMPLIST* list.

Notice, also, that the *display\_mode* property is set to *\$tsml\_col\_mode*. This is a special XSLT variable used inside the grid to indicate the display mode defined in the *DEF\_LIST*.

Now an *on\_change* action will be added to the combo box to execute a

Webroutine when a value is changed. The Webroutine, UpdateDepartment, takes two inputs: *EMPNO* and *DEPARTMENT*.

6. Set the *on\_change\_wrname* property of the combo box to **UpdateDepartment**.
7. Set the *tagfield1* property of the combo box to **DEPARTMENT**. This tells the combo box to submit the selected value of the *DEPARTMENT* column (in the *DEPTS* list).
8. Set the *reentryfield* property of the combo box to **EMPNO** and set the *reentryvalue* property to **../xml:column[@name='EMPNO']**. As this is an XPath expression you must use the XPath entry area at the bottom of the details tab to enter the value. This tells the combo box to get the value of the *EMPNO* column of the grid and submit it in a field called *EMPNO*.

### **8.1.13 Image (std\_image)**

[QuickStart - Image Properties – Image \(std\\_image\)](#)

Displays an image. Has the option to load the image only when it comes into view, which helps render the page faster.

## **QuickStart - Image**

To use an image:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Standard Image weblet.
2. Set the `relativeImagePath` property to the image you want to display.

## Properties – Image (std\_image)

caption hideIf relativeImagePath  
height lazyLoad width

## **relativeImagePath**

The path and file name, relative to the images virtual directory, of the image to be displayed.

### **Default value**

Blank – no image is displayed.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **lazyLoad**

If true, the image is not loaded until it comes into view

### **Default value**

True.

### **Valid values**

true(), false() or any valid XPath expression that returns a boolean value.

## **width**

Specifies the width of the input in pixels.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

### **Default value**

Blank - the browser will calculate the width after it loads the image.

### **Valid values**

Any integer value.

## **height**

Specifies the height of the input in pixels.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

### **Default value**

Blank - the browser will calculate the height after it loads the image.

### **Valid values**

Any integer value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**caption**

Specifies alternate text for the user, if he/she for some reason cannot view the image (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

**Default value**

Blank

**Valid values**

Any string value.

### 8.1.14 List Paging Images (std\_list\_images) and List Paging Buttons (std\_list\_buttons)

[QuickStart - List Paging Images & List Paging Buttons](#)

[Properties - List Paging Images & List Paging Buttons](#)

The List Paging Images weblet includes three images for navigating to the previous page, next page and starting a new search. This weblet is designed to be included before or after a list visualization with supporting logic for page at a time processing of list information.

The appearance of the Previous and Next images can be conditioned.

The weblet looks like this:



and is usually implemented something like this:

Dept Code	Department Description
ABCD	ABCDEFGHIJKLMNOPS



The List Paging Buttons weblet is similar but uses push buttons instead of images for navigation.

## QuickStart - List Paging Images & List Paging Buttons

To use either of these webllets you would create a webroutine that specifies a working list in its WEB\_MAP. The WAM should be designed for page at a time processing with a single page of entries passed in the defined working list. The working list will be visualized as a table on the resulting web page and by adding a row to the table you can incorporate navigation by completing the following steps in the LANSA Editor:

1. Add a row to the table generated to show the working list To do this, right-click in the list and select Table – Add Row - 1 from the pop-up menu.
2. If the table includes more than one column, focus on the first column in the newly created row and click on the Design tab Set the colspan property to the number of visible columns in the table This will allow the navigation weblet to display across the full width of the table.
3. Click on the Webllets tab, select Standard Webllets from the drop-down list near the top and locate the List Paging Images (or List Paging Buttons) weblet.
4. Drag and drop into the newly created table row in the Design view Make sure the weblet is selected and then click on the Details tab.
5. Set the on\_page\_wname property to the webroutine to be invoked when either the Previous or Next image (buttons) is clicked.
6. Set the on\_search\_wname property to the webroutine to be invoked when the Search image (button) is clicked.
7. Set an appropriate field name in the reentryfield property or use the default field STDREENTRY This field is required to distinguish between previous and next page processing in the RDMLX code.

## Properties - List Paging Images & List Paging Buttons

The List Paging Images & Buttons Weblet's properties are:

<code>class</code> (std_list_buttons only)	<code>nextcondfield</code>	<code>protocol</code>
<code>formname</code>	<code>on_click_wamname</code>	<code>reentryfield</code>
<code>height_design</code>	<code>on_page_wrname</code>	<code>show_first_last</code>
<code>hide_if</code>	<code>on_search_wrname</code>	<code>tab_index</code>
<code>image_size</code>	<code>page_count_fieldname</code>	<code>vf_wamevents</code>
<code>mouseover_class</code>	<code>pos_absolute_design</code>	<code>width_design</code>
<code>name</code>	<code>prevcondfield</code>	

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the grid.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**image\_size**

The size of the weblet images in pixels. There are 4 sizes available: 16x16, 24x24, 32x32, 48x48.

**Default value**

16

**Valid Values**

16, 24, 32 or 48

## **prevcondfield**

The name of the field that determines whether the 'Previous Page' image / button is shown. A non-blank value displays the button.

### **Default value**

'STDPREV'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

## **nextcondfield**

The name of the field that determines whether the 'Next Page' image / button is shown. A non-blank value displays the button.

### **Default value**

'STDMORE'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**show\_first\_last**

A Boolean property. Set to true() to enable the *First* and *Last* buttons.

**Default value**

false() (that is, the *First* and *Last* buttons will not be shown)

**Valid values**

true(), false() or any valid expression, involving field names, literals or XSL variables, which can be resolved to true() or false().

## **reentryfield**

The field name to be used to indicate which button was selected when transfer is passed to the webroutine nominated in on\_page\_wname.

### **Button Re-entry Field Value**

First F

Previous P

Next M

Last L

### **Default value**

'STDREENTRY'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

## hide\_if

A Boolean property. An expression which if evaluated to be True will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

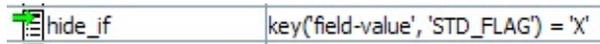
True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:



When the property loses focus, the expression is shown as follows:



**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

'0%' (this is equivalent to the weblet adopting its default width of 100%).

**Valid values**

A width in a valid unit of measurement in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height in a valid unit of measurement, in single quotes.

### **on\_click\_wamname**

The name of the WAM to be invoked when any of the images / buttons is clicked. This is used in combination with the on\_page\_wname and on\_search\_wname properties.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_page\_wrname**

The nominated webroutine will be invoked whenever the Previous or Next image / button is pressed.

**Default value**

Not applicable – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_search\_wrname**

The nominated webroutine will be invoked whenever the Search button is pressed.

**Default value**

Not applicable – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **page\_count\_fieldname**

The name of the field posted to the webroutine when the previous or next image / button is clicked that holds the number of items to show per page.

### **Default value**

Blank.

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**class**

*std\_list\_buttons only.*

The Cascading Style Sheet class to be applied to the buttons.

**Default value**

'STD\_BUTTON' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **mouseover\_class**

*std\_list\_buttons only.*

The Cascading Style Sheet class to be applied to the buttons when the mouse is moved over it.

### **Default value**

'STD\_BUTTON\_MOUSOVER' - The name of the shipped mouseover class for the weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

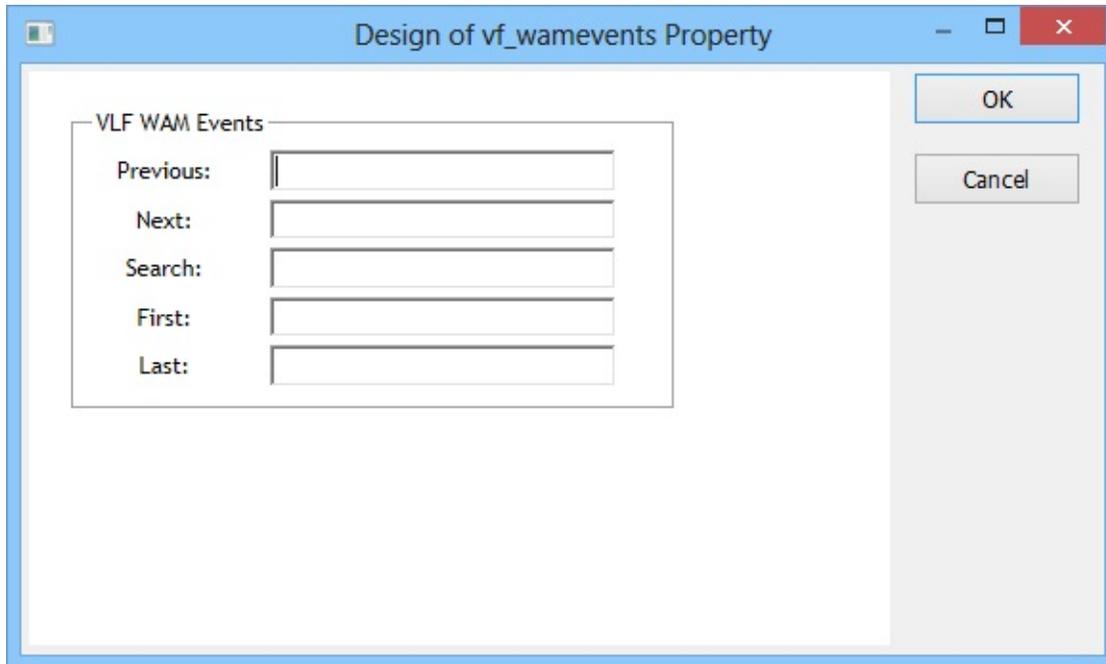
Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## vf\_wamevents

VLf WAM event strings for each action. Use the pop-up dialog to enter VLf WAM Event strings:



The image shows a Windows-style dialog box titled "Design of vf\_wamevents Property". It features a standard title bar with minimize, maximize, and close buttons. The main content area is titled "VLf WAM Events" and contains five text input fields, each preceded by a label: "Previous:", "Next:", "Search:", "First:", and "Last:". To the right of the input fields, there are two buttons: "OK" and "Cancel".

The First and Last VLf WAM events are disabled if the show\_first\_last property is False.

### Default value

Blanks.

### Valid values

Comma separated list of string values. Comma (',') not allowed in string value.

## 8.1.15 Mark-up (std\_markup)

### [QuickStart – Mark-up Properties – Mark-up](#)

The mark-up weblet is a companion weblet to the CKEditor Rich Text Editor when you want to visualize the content in output mode only.

#### Sample Mark-up



Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sed sollicitudin felis. Sed quis nunc nibh. Cras rutrum ornare condimentum. Sed bibendum, orci sed condimentum ultricies, magna massa congue lorem, et sagittis dui tellus sagittis nibh. Duis luctus nunc ac sapien mattis quis eleifend turpis adipiscing. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sed sollicitudin felis. Sed quis nunc nibh. Cras rutrum ornare condimentum. Sed bibendum, orci sed condimentum ultricies, magna massa congue lorem, et sagittis dui tellus sagittis nibh. Duis luctus nunc ac sapien mattis quis eleifend turpis adipiscing. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In sed sollicitudin felis. Sed quis nunc nibh. Cras rutrum ornare condimentum. Sed bibendum, orci sed condimentum ultricies, magna massa congue lorem, et sagittis dui tellus sagittis nibh. Duis luctus nunc ac sapien mattis quis eleifend turpis adipiscing.

As the mark-up content can have script elements and input fields, you should use the mark-up weblet with care, only with content that you can trust or that has been verified.

## **QuickStart – Mark-up**

To use the mark-up weblet you can follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Mark-up weblet.
2. Drag and drop the weblet onto your page. Make sure the weblet is selected and then click on the Details tab. Change the name to the field name that you want to visualize as marked-up content.

## Properties – Mark-up

class	name	value
height	pos_absolute	valueFromField
hide_if	title	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the mark-up weblet.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **valueFromField**

If set to true and the value property is null, the Mark-up weblet will load the value from the field that matches the Mark-up name attribute. Use this option if your text content is large and you don't want the content to appear both in the Mark-up weblet value and the webroutine field values list.

### **Default value**

False()

### **Valid values**

Any valid XPath expression that returns a Boolean value.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

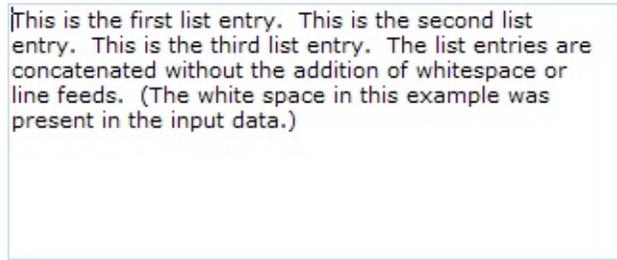
### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.1.16 Memo using a list (std\_list\_textarea\_v2)

[QuickStart - Memo using a list](#) [Properties - Memo using a list](#)

The Memo using a list (text area) weblet provides a text area for the display and input of long text values, possibly spanning multiple lines. It broadly corresponds to the <textarea> html element. The weblet looks like this:



This is the first list entry. This is the second list entry. This is the third list entry. The list entries are concatenated without the addition of whitespace or line feeds. (The white space in this example was present in the input data.)

This weblet is similar to the Memo using a field (std\_textarea) weblet. The difference concerns the means by which the text is exchanged between the page and the webroutine. In this weblet, the text is exchanged using a working list – refer to the description of the listname property for further information.

## QuickStart - Memo using a list

To use this weblet you would typically create a webroutine that specifies a working list in its WEB\_MAP that contains and populates one text field. When LANSA generates the default XSL for such a webroutine, the list will usually be visualized in a table. To use this weblet in place of the table, follow these steps:

1. Delete the table that visualizes the list.

To do this, right-click in the list and select Delete Entire List from the pop-up menu.

If the list has been specified with the \*hidden WEB\_MAP attribute, it must be manually removed from the XSL (not visualized in Design View).

2. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Memo using a list weblet.
3. Drag and drop the weblet onto your page in the Design view Make sure the weblet is selected and then click on the Details tab.
4. Set the listname and list\_text\_fieldname properties as required to refer to the appropriate field in your working list.
5. Set the cols property appropriately according to the size of the field in your application.

## Properties - Memo using a list

The Memo using a list weblet's properties are:

class	list_text_fieldname	read_only
cols	listname	rows
disabled	max_rows_onsubmit	tab_index
formname	name	width_design
height_design	onchange_script	word_wrap_display
hide_if	pos_absolute	word_wrap_onsubmit

**listname**

The name of the working list to use to populate the text in the textarea and/or receive text from the text area. The list should be specified in the WEB\_MAP for the webroutine. The list\_text\_fieldname property identifies the field in the working list that contains the text.

Upon output, the rows from the working list are concatenated to form the displayed text. Each row in the working list is a new line in the text area.

Upon input, the text in the text area is broken into strings written to the working list. The exact mechanism used will depend on the values of the cols and word\_wrap\_onsubmit properties.

**Default value**

No default value applies. You must specify the list name.

**Valid values**

The name of the working list, in single quotes. A list of known list names is available by clicking the corresponding dropdown button in the property sheet.

## **list\_text\_fieldname**

The name of the field in the working list specified by the listname property that is used to populate the text in the textarea and/or receive text from the text area. Refer to the listname property for more information on how the working list is used with this weblet.

### **Default value**

If the field name is not specified, the first field in the working list is assumed.

### **Valid values**

The name of the field, in single quotes. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

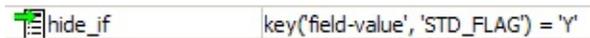
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered in this form:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is '#STD\_FLAG = 'Y''. To the left of the text is a small icon representing a document or list.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. To the left of the text is a small icon representing a document or list.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width\_design**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSA Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width – determined by the cols property).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height\_design**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height – determined by the rows property).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

**rows**

The number of visible rows in the textarea. This property sets the height of the weblet such that the specified number of rows or lines of text will be visible.

If the height\_design property is specified, it takes precedence and the rows property is ignored.

**Default value**

10

**Valid values**

A number that specifies the number of rows.

## **cols**

The number of columns in the textarea. This property sets the width of the weblet, based on the average character width for the font used, such that approximately the specified number of columns of text will be visible. Note that the mechanism used to determine the actual width is browser specific and will result in slightly different sizes on different browsers.

If the `width_design` property is specified, it takes precedence and the `cols` property is ignored.

### **Default value**

50

### **Valid values**

A number that specifies the number of columns.

## **word\_wrap\_display**

This property specifies how the weblet should handle word-wrapping of the displayed text. If true, long lines of text are automatically wrapped at the right edge of the text area. Note that this is display only. Wrapping of the text entered in the working list is determined by the `word_wrap_onsubmit` property.

### **Default value**

`true()`

### **Valid values**

Boolean values `true()` or `false()` or any expression that evaluates to true or false.

## **word\_wrap\_onsubmit**

This property specifies how the weblet should handle word-wrapping when submitting the text. If true, long lines of text are automatically wrapped at a width of cols characters. If a fixed-width font is used for the text area then this wrapping will match what the user sees on screen.

If false, long lines of text will not be wrapped.

Note that, to avoid data loss, word wrapping will still occur at the maximum width of the list\_text\_fieldname field regardless of the value of this property.

### **Default value**

false()

### **Valid values**

Boolean values true() or false() or any expression that evaluates to true or false.

**max\_rows\_onsubmit**

The maximum number of rows to submit. If your working list has a maximum size, you should set this property to that value.

**Default value**

0 (which means to send all rows)

**Valid values**

A number that specifies the number of rows.

**class**

The Cascading Style Sheet (CSS) class name for the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

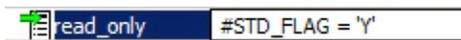
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**onchange\_script**

JavaScript code to be run when the text area loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

**Default value**

Blank. No JavaScript is run.

**Valid values**

Any valid JavaScript statement(s).

## 8.1.17 Large List (std\_largelist)

### QuickStart - Large List Properties - Large List

The Large List weblet can be used to display large lists. This weblet is suitable for report-like, output-only lists that don't require special formatting. The list can be sent to the user-agent either as an XHTML list or as a CSV (Comma-separated values) file.

Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ
EfGhIjKlMnOp	3,456	AbCdEfGhIjK	87,654,321	AbCdEdGhIjK	2007-12-31	LmNoPqRsT	4795439	EfGhIjKlMnOpQrSt	IjKlMnOpQrStUv
AbCdEf	123,456	AbCdEfGhIjKlMnOpQrStU	987,654,321	VwXyZ	2006-06-24	VwXyZ	1654789	CdEfGhIjKlMnOp	RsTuVwXyZ

## QuickStart - Large List

1. Create your (main) webroutine as you would normally do. However, don't define a WEB\_MAP for your working list. Instead, include the Large List weblet where you would place the working list.
2. Create a separate webroutine to serve the large list. Define a WEB\_MAP FOR(\*OUTPUT) for the working list. Define WEB\_MAP FOR(\*INPUT) for fields you need to tailor the content of your list.
3. In your main webroutine, edit the Large List weblet properties:
  - a. Nominate the webroutine you created in step 2 as the webroutine to serve the list.
  - b. In field\_names\_to\_exchange, select the fields you want to make available to the webroutine serving the list.
4. To view the list, run your main webroutine.

## Properties - Large List

The Large List weblet's properties are:

<code>column_css_class</code>	<code>format_target</code>	<code>show_busybox</code>
<code>csv_hyperlink_relative_image_path</code>	<code>iframe_height</code>	<code>hourMax</code>
<code>csv_hyperlink_text</code>	<code>iframe_width</code>	<code>src_wname</code>
<code>csv_hyperlink_type</code>	<code>listname</code>	<code>wait_content</code>
<code>fields_names_to_exchange</code>	<code>name</code>	

**name**

The name of the list. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the grid.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text. Must be unique within the whole XHTML page.

**listname**

The name of the working list to use to populate the large list.

**Default value**

Blank.

**Valid values**

Single-quoted text. The list must be mapped for output in the webroutine that serves the list (See `src_wname` property).

## **format\_target**

The format/target of the data served by the source WAM webroutine.

### **Default value**

xhtml-iframe

### **Valid values**

1. xhtml-iframe: The list is formatted as an XHTML table inside an iframe. Use this option if you want users to select the whole table using Ctrl+A (Select All) when inside the iframe
2. xhtml-inline: The list is formatted as an XHTML table and placed inline in the main XHTML page. This is the closest to having the XHTML defined in the main XHTML page.
3. csv-inline: The list is formatted as a comma-separated document inside an iframe. If your browser is Microsoft Internet Explorer and you have Microsoft Excel, the iframe will show the list as a spreadsheet.
4. csv-window: The list is formatted as a comma-separated document and opened in a new window. If your browser is Microsoft Internet Explorer and you have Microsoft Excel, the new window will show the list as a spreadsheet.

### **Warning:**

- The xhtml-inline format/target loads the list programmatically. The page cannot be saved. If you need to save the page, use the xhtml-iframe option (You can save the iframe content).

## **iframe\_width**

The iframe width. (Only applicable when the content target is an iframe.)

### **Default value**

'auto'

### **Valid values**

Enter the width as a quoted literal (You can use percentages, points or pixels as units). If your format/target is 'xhtml-iframe' you can use the special value 'auto' to resize the iframe so the contents are visible without a horizontal scroll bar.

## **iframe\_height**

The iframe height. (Only applicable when the content target is an iframe.)

### **Default value**

'auto'

### **Valid values**

Enter the height as a quoted literal (You can use percentages, points or pixels as units). If your format/target is 'xhtml-iframe' you can use the special value 'auto' to resize the iframe so the contents are visible without a vertical scroll bar.

## **column\_css\_class**

Boolean value. Only applicable if the format/target is XHTML. If true, a cascading style sheet (CSS) class is assigned to each column in the XHTML table. If false, a cascading stylesheet class is assigned only at the row level.

### **Default value**

false(). Apply CSS style at the row level.

### **Valid values**

true(), false() or an XSLT expression that returns a boolean result.

#### **CSS styles for Large Lists**

The preformatted XHTML table has the CSS class "std\_largelist". By creating a CSS style for this class or its sub elements, you can customize the look of your Large List.

When the target is an iframe, the iframe inherits the styles from the parent window (the main webroutine).

For example, if your webroutine overrides the styles as follows:

```
table.std_largelist th, table.std_largelist td {white-space:nowrap;}  
tr.list-h {background:black; color:white; font-weight:bold;}  
table.std_largelist tr.list-o {background:white; color:black;}  
table.std_largelist tr.list-e {background:#ffe9bd; color:black;}
```

your list will look like:



**src\_wamname**

Specifies the name of the WAM whose webroutine is executed to serve the list. (The webroutine name is specified in the src\_wname property.)

**Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName)

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**src\_wrname**

Specifies the name of the webroutine that is executed to serve the list. (The name of the WAM containing the webroutine is specified in the src\_wamname property.)

**Default value**

No default value. You must enter a valid webroutine name.

**Valid values**

The name of a Webroutine in single quotes. The webroutine must exist in the WAM specified in the src\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **fields\_names\_to\_exchange**

Optional. Select one or more fields to send to the webroutine serving the list. You would normally nominate fields whose values will determine the content of the list. To select more than one field, press the Ctrl key while selecting additional field names.

The fields are sent to the webroutine serving the list but are not read back into the main webroutine. That is, the behavior is identical to having them mapped for input in the webroutine serving the list.

### **Default value**

No default value.

### **Valid values**

Comma separated list of field names mapped for output in the current webroutine.

## **csv\_hyperlink\_type**

Specifies whether to show the hyperlink to the new CSV window as an image or a text link. Only applicable for format/target 'csv-window'.

### **Default value**

image.

### **Valid values**

image: The hyperlink is shown as an image. You define the image in the `csv_hyperlink_relative_image_path` property.

text: The hyperlink is shown as a text link. You specify the text in the `csv_hyperlink_text` property

## **csv\_hyperlink\_relative\_image\_path**

The path and name, relative to the images directory, of the image to be displayed to represent the hyperlink to the new CSV window. Only applicable for format/target 'csv-window'.

### **Default value**

'excel.gif' which is an image representing a Microsoft Excel spreadsheet.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **csv\_hyperlink\_text**

The text for the hyperlink to the new CSV window. This is the hyperlink text when you choose 'text' in the csv\_hyperlink\_type property. Only applicable for format/target 'csv-window'.

### **Default value**

'Hyperlink'

### **Valid values**

Any appropriate literal, field name, system variable name or multilingual text variable name. A field, system variable or multilingual variable name can be chosen from a list by clicking the corresponding ellipses button in the property sheet.

## **show\_busybox**

Choose whether to show a busy box message while the list is being served. This is recommended for large lists where it can take significant time to prepare, send and render the list.

The busy box message doesn't work with new windows. It only works for inline content or content loaded into an iframe.

### **Default value**

true(). The busy box message is shown.

### **Valid values**

true(), false() or an XSLT expression that returns a boolean result.

## **wait\_content**

Optional. The message to be displayed in the busy box while the large list is being served. Only applicable if the show\_busybox property is true().

### **Default value**

Blank – the default 'Processing' text is shown.

### **Valid values**

Any appropriate literal, field name, system variable name or multilingual text variable name. A field, system variable or multilingual variable name can be chosen from a list by clicking the corresponding ellipses button in the property sheet. Keep the message short.

### **Example**

This example shows a literal message:



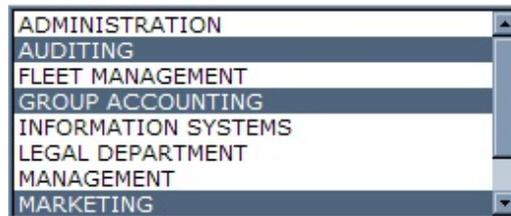
The busy box will display:



## 8.1.18 List Box (std\_listbox)

### [QuickStart - List Box Properties - List Box](#)

The List Box weblet provides a Windows-like single- or multi-select list box for your web page. It looks like this:



## **QuickStart - List Box**

Refer to the example in List Box Example.

## Properties - List Box

The List Box weblet's properties are:

allow_multi_selections	listname	reentryvalue
captionfield	mouseover_class	selector_field
class	multi_select_codefield	selector_value_eq
codefield	multi_select_listname	size
disabled	name	submit_tagfields
display_mode	on_select_wamname	tab_index
formname	on_select_wrname	tagfield1, tagfield2, tagfield3
height_design	pos_absolute	target_window_name
hide_if	protocol	value
items	reentryfield	vf_wamevent
		width_design

**name**

The name of the list box. If the list box is to allow selection of only one entry, this should be the name of the code field, in quotes (for example, 'DEPARTMENT'), that is to contain the value of the selected entry to be returned to the WAM. If it's to be multi-select, its name can remain as the default.

**Default value**

concat('o', position(), '\_LANSA\_n') – this is the internal name given to the weblet by LANSA.

**Valid values**

A name, in single quotes. This can be a nominal choice or the name of a field, the value of which is set by the weblet.

**value**

The value to set the weblet to. For a single select list, this should be the name of the field containing the code value to be pre-selected (for example, \$DEPARTMENT). For a multi-select list, this property can be left at its default.

**Default value**

Blank.

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

**display\_mode**

Controls whether the weblet accepts input or displays output.

**Default value**

'input'

**Valid values**

'input' or 'output'.

## items

An XML node set specifying the items to appear in the weblet. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. Leave blank if items are populated from a list specified in the listname property.

### Default value

`document(")*/lxml:data/lxml:listbox` – this indicates no items have been defined for this list box.

### Valid values

Not applicable – this value is system maintained. To invoke the designer, use the ellipse button in the property sheet.

### Example

This example indicates that items have been setup in the designer to use as list box values.



```
items document(")*/lxml:data/lxml:caption_value_pairs[@id='4A14173FA6D64A059507F9124A630CF3']
```

Using the ellipses button on the property you will see the designer and be able to maintain the items to be displayed in the list box. The following view of the designer shows two entries for the list box. The first entry has the literal value 'MONDAY' and the second entry uses a multilingual variable to display the description for the code TUE. Check the Default Item check box for the item which is to be selected if no value is preselected.

Design of items Property

Items

- MONDAY
- \*MTXTDEMCALEN06001

Move Up

Move Down

Add New

Remove (DEL)

Item Properties

Caption:  Or \*MTXT Variable: \*MTXTDEMCALEN06001 ...

Value:   Default Item

Selector Value:

OK

Cancel

Tip: Select an item in the list to edit it.

**size**

The number of visible rows in the list. This property will be ignored if the height\_design property is specified.

**Default value**

8

**Valid values**

A valid number.

## **allow\_multi\_selections**

A Boolean property that controls whether multiple selections are allowed in the list box. If multiple selections are allowed, the `multi_select_listname` and `multi_select_codefield` properties must be specified.

### **Default value**

`false()` – only a single selection may be made in the list box.

### **Valid values**

`True()`, `false()`, or a valid expression that returns True or False.

## **multi\_select\_listname**

The working list to contain the selected entries for the list box if the `allow_multi_selections` property is True. The working list should contain the code field that is specified in the `multi_select_codefield` property.

### **Default value**

Blank – only a single selection may be made in the list box.

### **Valid values**

The name of a working list, in single quotes. Click the corresponding dropdown button in the property sheet to choose from a list of known working lists.

## **multi\_select\_codefield**

The name of the field in the working list specified in the multi\_select\_listname property that will hold the code value of the selected list box items.

### **Default value**

Blank – only a single selection may be made in the list box.

### **Valid values**

The name of a field, in single quotes. Click the corresponding dropdown button in the property sheet to choose from a list of known fields.

**listname**

The name of the working list that contains the items used to populate the weblet.

**Default value**

Blank. A valid list name must be entered.

**Valid values**

The name of a valid working list, in single quotes. A list of valid list names can be chosen from by clicking the corresponding dropdown button in the property sheet.

**selector\_field**

The name of the field in the list specified in the listname property that can contain a value to limit, to a subset, the list items shown in the weblet.

**Default value**

Blank. All entries of the list are always shown.

**Valid values**

The name of a valid field, in single quotes. A list of valid field names can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **selector\_value\_eq**

The value that the field specified in the selector\_field property must be equal to in order to limit, to a subset, the list items shown in the weblet. If list details have been entered using the items property this value must match the selector value .

## **Default value**

Blank. All entries of the list are always shown.

## **Valid values**

A valid value for the type of field specified in the selector\_field property or value enter in items property.

**codefield**

The name of the field in the list specified in the listname property that holds the key value for each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**captionfield**

The name of the field in the list specified in the listname property that holds the caption for the each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

### **tagfield1, tagfield2, tagfield3**

The name of a field in the list specified in the listname property that can contain an additional value to be tagged onto a list item. This value is added as an attribute with the name of the field prefixed with 'tag\_'. Note that the value is not shown in the list box, but can be referenced in JavaScript code.

#### **Default value**

Blank.

#### **Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **submit\_tagfields**

A Boolean property that controls whether the values in the fields specified in the tagfieldn properties are submitted to the web server when submitting the form.

### **Default value**

True() – the values in the fields are posted to the web server.

### **Valid values**

true(), false(), or a valid expression that returns true() or false().

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

### **Default value**

'STDREENTRY'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

true(), false(), or a valid expression that returns true() or false().

### Example

This example will hide the weblet if field STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**on\_select\_wamname**

The name of the WAM to be invoked when an item in the list box is selected.

**Default value**

Blank – this equivalent to the current WAM.

**Valid values**

A WAM name, in single quotes. Click on the corresponding dropdown button in the property sheet to choose from a list of known WAMs.

**on\_select\_wrname**

The name of the Webroutine to be invoked when an item in the list box is selected.

**Default value**

Blank – a Webroutine is not invoked when an item is selected.

**Valid values**

A Webroutine name, in single quotes. This Webroutine should exist in the WAM specified in the on\_select\_wamname property. Click on the corresponding dropdown button in the property sheet to choose from a list of known Webroutines.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false(), or a valid expression that returns true() or false().

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_listbox' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**mouseover\_class**

The Cascading Style Sheet class name of the weblet when the mouse is moved over it.

**Default value**

Blank.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## List Box Example

Following is an example of a list box with multiple selections plus a description of the techniques used to produce this result.

If you require only single selection in your list box, you should refer to the description of the [allow\\_multi\\_selections](#) and related properties.



The list box allows for multiple selections (via the standard Windows method of holding down the Shift or Ctrl keys when making selections). The 'Produce Report' button, when clicked, passes the selected entries into the WAM for processing.

Change the items selected in the list box and click the 'Produce Report' button. You will see a list of the selected entries in a list, similar to the following:



## The List Definitions

Two lists are required to code the multi-select list box:

```
Def_List Name(#ListBox) Fields(#DEPARTMENT #DEPTDESC) Type(*Working)
Def_List Name(#Selected) Fields(#DEPARTMENT) Type(*Working)
```

The first, #ListBox, drives the content of the list box. It will usually have two fields in it: a code field and a description field.

The second list, #Selected, has two uses. Entries can be added to it in the WAM to cause #ListBox entries to be pre-selected when the list box is displayed. It is also used to return the entries selected by the user to the WAM.

If you right-click on the ShowPage Webroutine and open it in the LANSAsoft Editor, you'll see the list box and the push button. Click on the list box and display its properties by clicking on the Details tab:

allow_multi_selections	True
multi_select_listname	SELECTED
multi_select_codefield	DEPARTMENT
listname	LISTBOX
selector_field	
selector_value_eq	
codefield	DEPARTMENT
captionfield	DEPTDESC

The allow\_multi\_selections setting is self-explanatory: true() means that multi-selections are enabled, whilst false() ensures that only single selection is

permitted.

- The listname property holds the name of the working list used to populate the list box.
- The codefield property is the name of the field in the listname list that holds the code value of the list entry (in this case, department code).
- The captionfield property is the name of the field to be displayed in the list box (in this case, department description).

These entries are sufficient to get the list box populated and onto the web page. The remaining two properties relate to the selection of entries in the list box:

- The multi\_select\_listname property contains the name of the list that is to return the selected entries to the WAM.
- The multi\_select\_codefield property is the name of the field in that list that is to hold the code value of the selected entry.

## Building the Main List

Look at the ShowPage Webroutine in the source editor again:

```
Webroutine Name(ShowPage)
  Web_Map For(*output) Fields(#ListBox *private) (#Selected *private)

  Select Fields(*ALL) From_File(deptab)
    Add_Entry To_List(#ListBox)
  Endselect

  #DEPARTMENT := ADM
  Add_Entry To_List(#Selected)

  #DEPARTMENT := INF
  Add_Entry To_List(#Selected)

Endroutine
```

Note the Web\_maps for the two list box-related lists.

A simple Select loop is used to add all departments from the DEPTAB file to the main list, #ListBox.

Then, two entries are added to the #Selected list. This will pre-select the Administration and Information Systems departments in the list box when it's displayed.

## Processing the Selected Entries

The ProduceReport Webroutine is invoked by the push button on the web page:

```
Webroutine Name(ProduceReport)
  Web_Map For(*input) Fields(#Selected)
  Web_Map For(*output) Fields(#Report)

  Def_List Name(#Report) Fields(#DEPARTMENT) Type(*Working)

  Selectlist Named(#Selected)
    Add_Entry To_List(#Report)
  Endselect
Endroutine
```

Again, note the Web\_maps: the incoming list of selected entries (#Selected) and another, outgoing list (#Report). These are used to display a list of selected departments.

## 8.1.19 Menu bar (std\_menubar)

[Using the Menu Item Designer](#) [Using a List to Define the Menu](#)  
[QuickStart - Menubar](#) [Properties - Menu bar](#)

The menu bar weblet provides the functionality of a menu bar that can invoke other web pages including other web routines. The menu bar can be arranged horizontally or vertically and the top level menu items can cause further menus to pop-up as the mouse moves over them. This is what the menu bar weblet looks like when arranged horizontally – in this example, two levels of popup menu are shown.



The menu items can be defined in the Webroutine design using the menu item designer or they can be supplied at runtime in an RDMLX list. The weblet is based on the jQuery UI menu widget and requires jQuery and jQuery UI to operate (the weblet will automatically add the required external resources to the output HTML).

## **QuickStart - Menubar**

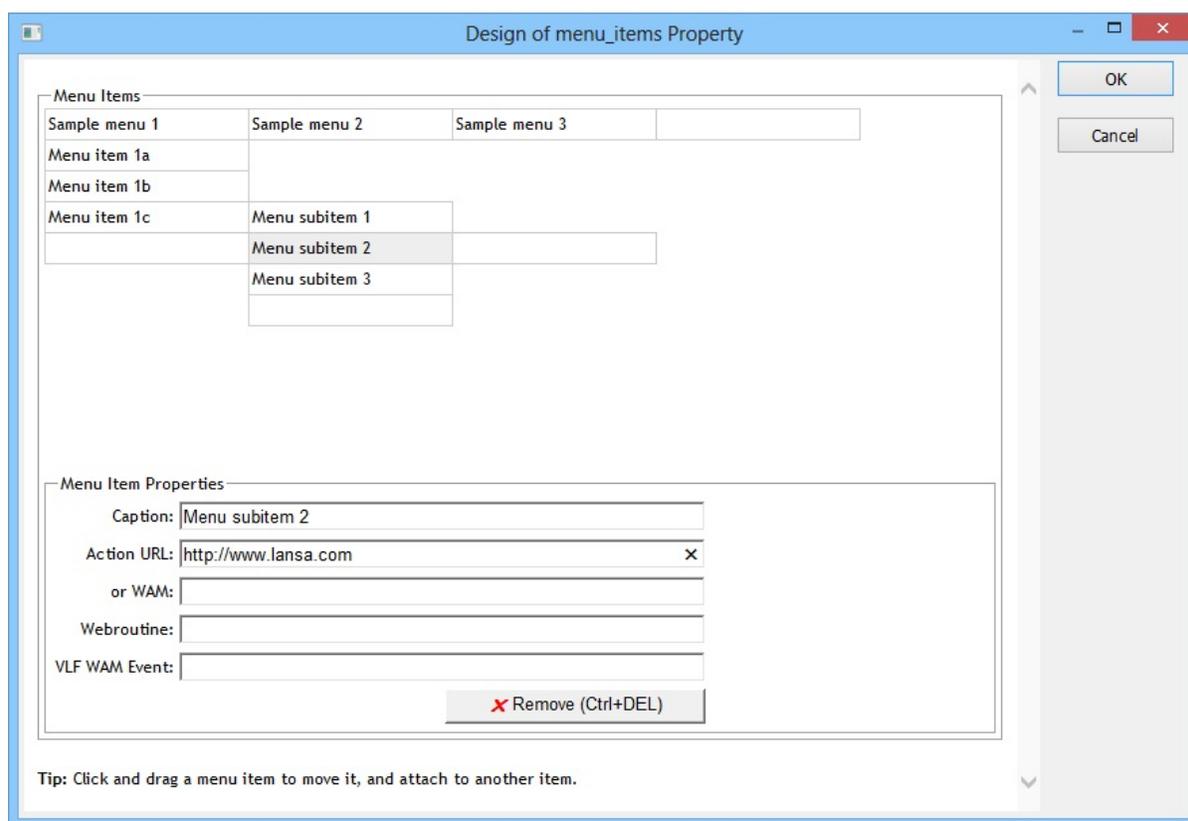
To use the Menubar weblet, open your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Menubar weblet.
2. Drag and drop the weblet onto your page in the Design view. Make sure the weblet on the page is selected and then click on the Details tab.
3. Click the ellipsis button next to the menu\_items property to open the menu items designer and define your menu items.

## Using the Menu Item Designer

To open the menu item designer, follow these steps:

1. Make sure the Menubar weblet on your page is selected and then click on the Details tab.
2. Move the mouse pointer over the menu\_items property in the Details tab. An ellipsis should appear next to the property value. Click the ellipsis button to open the menu items designer. A window like the one shown below appears.



The top-half of the window shows a representation of the current state of the menu. Note that this is always shown in horizontal orientation irrespective of the current value of the orientation property. In this part of the window, you can:

- click on items to complete their details in the lower half;
- drag and drop items to rearrange them;
- add new items by clicking on the blank items shown adjacent to the currently selected item;
- remove items by selecting them and clicking the Remove button.

In the bottom-half of the window you can specify the details for the selected

item as follows:

**Caption**

Specifies the text that appears on the face of the menu item. You can also enter the text directly on the face of a menu item by clicking on it in the top-half of the window and typing.

**Action URL**

Specifies a URL that the menu item will navigate to when clicked. You can specify a complete URL or one that is relative to the current page.

**WAM and Webroutine**

Specifies a Webroutine to invoke when the menu item is clicked. Note that if both a URL and a Webroutine have been specified, the URL will take priority and the Webroutine will be ignored.

**VLF WAM Event**

Specifies a VLF WAM event to pass to the handler when this menu item is selected.

## Using a List to Define the Menu

The menu structure can be defined with an RDMLX list (normal list or JSON list). To do this you must create a list with a specific format. The list must have 6 columns containing the following values:

1. **Menu item ID** - A unique id for the menu item. This can be a number or a string. If using numbers, do not use zero (items with an id of zero cannot have child items).
  2. **Parent item ID** - The id of the parent item. Use zero (0) or an empty string for top level items with no parent.
  3. **Caption** - The text to display for the menu item.
  4. **URL** - The URL that the menu item will navigate to when clicked. You can specify a complete URL or one that is relative to the current page. Use an empty string if the menu item is to invoke a webroutine.
  5. **WAM** - The name of the WAM containing the Webroutine to be invoked when the menu item is clicked. If the URL is specified, this is ignored.
  6. **Webroutine** - The name of the Webroutine to be invoked when the menu item is clicked. If the URL is specified, this is ignored.
  7. **WAM Event (optional)** – The VLF WAM Event to send to the webroutine handling the request.
- The column names may be anything you like but they must be defined in the order specified above.

## Properties - Menu bar

The Menu bar weblet's properties are:

id          menu\_items show\_arrows  
listname orientation submit\_selected\_to

## **id**

A unique name for the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

### **Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

### **Valid values**

Any valid HTML ID string. It must begin with a letter ([A-Za-z]) and may be followed by any number of letters, digits ([0-9]), hyphens ("-"), underscores ("\_"), colons (":"), and periods ("."). However, for best compatibility with various JavaScript libraries, CSS editors and other web technologies it is advisable to avoid colons (":") and periods (".").

**listname**

The name of an RDMLX list that contains the definition of the menubar and its items. See [Using a List to Define the Menu](#) for more details of how to use this property. Both normal lists and JSON lists are allowed. JSON lists are recommended.

**Default value**

Blank.

**Valid values**

Any output list from the current Webroutine.

## **menu\_items**

An XML nodeset that specifies the menu items. This is a system generated value set up when you drag the menu onto the design view.

Do not directly edit the value shown. Instead, click the ellipsis button to open the menu items designer. Refer to [Using the Menu Item Designer](#) for more information.

### **Default value**

`document(")/*/lxml:data/lxml:menu[@id='<unique id>']`

where the <unique id> is an automatically generated identifier.

### **Valid values**

Not Applicable. (This value is system generated and should not be modified.) If [listname](#) is specified, this value will be ignored.

## **orientation**

The orientation of the menu. This determines the positioning of the top-level menu items relative to each other and the direction or relative location that pop-up menus appear.

### **Default value**

'top'

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

'top' The top-level menu items are arranged horizontally and first-level pop-up menus appear below the corresponding top-level menu item. Suitable for use as a horizontal menu bar across or near the top of the page.

'left' The top-level menu items are arranged vertically and first-level pop-up menus appear to the right of the corresponding top-level menu item. Suitable for use as a vertical menu bar on the left of the page.

'right' The top-level menu items are arranged vertically and first-level pop-up menus appear to the left of the corresponding top-level menu item. Suitable for use as a vertical menu bar on the right of the page.

'bottom' The top-level menu items are arranged horizontally and first-level pop-up menus appear above the corresponding top-level menu item. Suitable for use as a horizontal menu bar across or near the bottom of the page.

**show\_arrows**

A boolean value that indicates whether to not to display arrows on the top level menu items. This only applies to the top level menubar items. Any menu items that have submenus will always display an arrow to indicate this fact.

**Default value**

true()

**Valid values**

true(), false() or a valid boolean expression.

## **submit\_selected\_to**

When a menu item is clicked, and the action of that menu item is to invoke a Webroutine, the ID of the selected menu item will be placed into this field. For menubars defined with an RDMLX list, the ID will be the ID specified in column 1. For menubars specified with the menu item designer, the value will be a comma delimited sequence of numbers representing the path to the selected item. For example, if the 3rd item in the second menu is selected, the submitted ID would be "2,3".

### **Default value**

Blank

### **Valid values**

The name of a \*INPUT field in the webroutine(s) being executed.

## 8.1.20 Menu item (std\_menu\_item\_v2)

[QuickStart - Menu item Properties - Menu item](#)

The menu item weblet provides a hyperlink menu item that looks and behaves very much like a push button. It is intended to be used in groups arranged together in the page to provide a menu of alternate pages or webrouines that the user can access from the current page.

The menu item is essentially a clickable link with some extra capabilities:

- It includes support for a "selected" style.

- It is constructed with several spans to allow for multiple images and expanding backgrounds (using the "sliding doors" technique).

## QuickStart - Menu item

The menu item weblet is used to provide a skeletal menu in the default standard webroutine layouts. You may wish to use them in your layouts to provide this functionality or you can use them directly in the page for individual webroutines. To use the menu item weblet, follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Menu item weblet.
2. Drag and drop the weblet onto your page in the Design view (You may wish to use a table or some other device to assist in arranging multiple instances of the weblet.) Click on the weblet and then click on the Details tab.
3. Set the caption property as required.
4. Set the `on_click_wrname` property to the name of the webroutine to be invoked when the hyperlink is clicked. If the webroutine is in a different WAM to the current webroutine then you will need to set the `on_click_wamname` property as well. Alternatively you can set the `href` property to cause the weblet to navigate to a destination that is not a webroutine.

## Menu Item Appearance

### Layout and Size

The basic menu item has no particular formatting beyond bold text and a CSS display value of "block". This can change significantly, depending on what layout and layout theme you are using. The default layouts provided prior to Visual LANSA V12 SP1 provided three themes: default, royal and grass.



Default

Royal

Grass

If you are using an new Layout with a jQuery UI theme then you can set the menu item to take on the standard theme appearance of a clickable item by setting the usejQueryUITheme property to True.

		
Cupertino	Smoothness	Pepper Grinder

If you have no theme applied, or have usejQueryUITheme set to False, the menu item will display as bold text inside a borderless white box

**Sample Menu Item**

The basic HTML structure of this is:

```
<a class="std_menu" id="webletName">  
<span class="std_menu_span1">  
<span class="std_menu_span2">  
Sample Menu Item  
</span>  
</span>
```

</a>

When a menu item is selected, the class of the <a> tag is changed to "std\_menu\_selected".

Basic styling of the menu item can be done by defining some properties for the std\_menu and std\_menu\_selected classes in your style sheet. For example:

```
a.std_menu, a.std_menu_selected {
  color:#01478c;
  border-style:solid;
  border-color:#7db0e5;
  border-width:0px 2px 1px 2px;
  background-color:#c7dff4;
}
a.std_menu_selected {
  background-color:#7db0e5;
}
```

A left aligned image could be added with something like this:

```
a.std_menu, a.std_menu_selected {
  background-image:url('icon.png') no-repeat left center;
  padding-left:32px;
}
```

More complex "self sizing" background images can be created by applying the two halves of the image to each of the spans in what is known as the "[sliding doors](#)" technique.

## Layout and Size

The default menu item is an HTML "block" item. This means that it automatically expands to the width of the space you give it and does not allow other content on either side of it (within the containing box). This is ideal for creating a vertical stack of menu items in a sidebar. You can set the size of the sidebar and the menu items will all automatically match the size. If you want to use the menu items in a different scenario, you may want to change their display mode and/or their size. For example, to create a horizontal bar of menu items like this:



You might place them inside a <div> with the id "hMenu" and then define the following CSS:

```
#hMenu {
background:#333;
}
a.std_menu, a.std_menu_selected {
color:#fff;
display:inline-block;
padding:5px 10px;
border-right:1px solid #fff;
background-color:#c7dff4;
font-weight:normal;
}
```

## Properties - Menu item

The Menu item weblet's properties are:

caption	is_selected_if_also	reentryvalue
class	name	selected_class
force_selected	on_click_wamname	style
formname	on_click_wrname	tab_index
hide_if	protocol	target_window_name
href	reentryfield	usejQueryUITheme
		vf_wamevent

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**caption**

This property specifies the text that is displayed for the menu item.

**Default value**

'Caption' (you will usually need to set this property value.)

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **href**

This property may be used to specify a URL that the menu item will navigate to. If specified, the URL may be specified as a literal value (for example 'http://www.mycompany.com/') or a field name may be specified that contains the URL at run-time.

This property takes precedence over the `on_click_wamname`, `on_click_wrname` and `protocol` properties. The latter properties are ignored if `href` is specified.

### **Default Value**

None

### **Valid Values**

A URL enclosed by single quotes or the name of a field, system variable or multilingual variable that will contain the URL at run-time.

### **on\_click\_wamname**

Specifies the name of the WAM whose webroutine is executed when the menu item is clicked. (The webroutine name is specified in the on\_click\_wname property.)

This property is ignored if the href property is specified.

### **Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName)

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

Specifies the name of the webroutine that is executed when the menu item is clicked. (The name of the WAM containing the webroutine is specified in the on\_click\_wamname property.)

This property is ignored if the href property is specified.

## **Default value**

No default value applies – either the href property or the on\_click\_wrname property must be specified in order for the menu item to be functional.

## **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

Typically you might use this property when it is necessary to switch to or from secure-mode processing. Otherwise it is not usually necessary to specify this property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. If specified, it is usually 'http:' or 'https:'.

## is\_selected\_if\_also

By default, a menu item assumes the selected-state appearance if the values for its `on_click_wamname` and `on_click_wrname` properties match the WAM and webroutine name of the current page (note that this comparison is case-sensitive).

This property allows you to extend this behavior by specifying a further condition that must apply for the menu item to assume the selected-state.

### Default value

`true()` (that is, no further condition applies)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example specifies that, in addition to the default tests specified above, the weblet will test if the field `#STD_FLAG` is equal to `'X'`. The expression should be entered in this form:



The screenshot shows a property editor for the `is_selected_if_also` property. The property name is on the left, and the value `#STD_FLAG = 'Y'` is entered in the text field on the right.

When the property loses focus, the expression is shown as follows:



The screenshot shows the same property editor after the text field has lost focus. The value `key('field-value', 'STD_FLAG') = 'Y'` is now displayed in the text field.

## **force\_selected**

If true, the menu item assumes the selected state regardless of the current WAM/Webroutine or the is\_selected\_if\_also property. If you want to apply your own logic to deciding the state of the menu item you can set is\_selected\_if\_also to False to disable the default logic and place an appropriate Boolean expression into force\_selected.

For example, you could create a field that contained the name of the selected menu item. Then you could set the force\_selected property (using the XPath entry area) to

```
#SELMENU = 'Menu1'
```

### **Default value**

False() (the default logic is applied)

### **Valid values**

Any valid XPath expression that returns a Boolean value.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

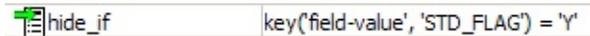
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is '#STD\_FLAG = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

## **target\_window\_name**

The name of the window, or frame, in which the destination of the menu item will be shown. A special value of "\_blank" will open a new unnamed window.

### **Default value**

Blank – the destination of the menu item will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

See the description of the reentryvalue property for further information.

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

No default value applies.

### **Valid values**

Single-quoted text.

## **reentryvalue**

The value to post to the target webroutine in the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the reentryfield property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. reentryfield: the field name that the target webroutine uses to refer to the information
2. reentryvalue: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** this property is provided to support a re-entrant programming technique that is commonly used in WEBEVENT applications. Web applications that are designed from the outset to use WAMs do not usually need to make use of that technique.

### **Default value**

No default value applies.

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **style**

The inline style property for the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

"

### **Valid values**

Any valid CSS properties, separated by semi-colons and contained within single quotes.

## **usejQueryUITheme**

Indicates if the weblet should apply jQuery UI classes to itself. If True, and the layout loads a jQuery UI theme, the layout will take on the look of a clickable element as defined by the theme.

### **Default value**

False

### **Valid Values**

Any valid XPath expression that returns a Boolean value.

**class**

The Cascading Style Sheet (CSS) class name of the menu item.

**Default value**

The name of the shipped class for the menu item.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**selected\_class**

The Cascading Style Sheet (CSS) class name of the menu item when it is selected.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **vf\_wamevent**

VLf WAM event string

### **Default value**

Blank.

### **Valid values**

String value. Comma (',') not allowed.

## **8.1.21 Navigation Panel (std\_nav\_panel)**

[QuickStart - Navigation Panel Properties - Navigation Panel](#)

The Navigation Panel weblet provides a container that can be used to display content that is provided by a WAM or a URL. As such, it can be used to 'modularize' your application such that not all content is provided by a single source.

## **QuickStart - Navigation Panel**

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Navigation Panel weblet.
2. Drag and drop it onto the web page being designed.
3. Set the nav\_wrname property to the name of the webroutine to be invoked when the web page is shown. If the webroutine is in a different WAM to the current webroutine then you will need to set the nav\_wamname property as well.

## Properties - Navigation Panel

The Navigation Panel weblet's properties are:

border	nav_wamname	transparent
border_width	nav_wrname	wait_content
class	pos_absolute	wait_content_absolute_imag
formname	protocol	wait_content_class
height	reentryfield	wait_content_image_alignn
hide_if	reentryvalue	wait_content_image_class
name	scrolling	wait_content_relative_imag
nav_asynchronously	size_panel_to_content	wait_content_timeout
nav_url	size_panel_to_content_axis	width
		vf_wamevent

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name in single quotes.

## **border**

The style of the border.

### **Default value**

Blank. The weblet has no border.

### **Valid values**

A valid border style, in single quotes. A valid style can be chosen from a list by clicking the corresponding dropdown button in the property sheet. Note that 'window-inset' is only supported by Internet Explorer.

**border\_width**

The width of the panel's border.

**Default value**

Blank. Equivalent to the panel having no border.

**Valid values**

A width, in a valid unit of measurement, in single quotes. Alternatively, a width of 'thick', 'medium' or 'thin' can be selected by clicking on the corresponding dropdown button in the property sheet.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

'100%' – the panel is sized to the width of the page.

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the weblet on the web page.

**Default value**

'250pt'.

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## **size\_panel\_to\_content**

A Boolean property that determines how the panel is sized. The navigated-to page must be in the same domain as the current page.

The size used will be determined by the browser and may vary slightly from browser to browser. It may also be affected by the initial size. For best results, make sure your panel content page explicitly sizes its content so that the browser doesn't have to decide for itself. Note, also, that Opera will not reduce the size of the panel.

If using *nav\_url*, the URL must be from the same server that the WAM is from. Browser security restrictions prevent code in a page from one server accessing information in a page from another server. If the panel content is from another server, the panel will not be able to determine its size. Using a URL that points to another server will cause the prompter code to fail with an "access denied" error.

### **Default value**

false() – the panel is not resized when navigated to.

### **Valid values**

true(), false(), 'once', or a valid expression. If set to true(), the frame will be sized to the content of the page it navigates to every time the navigation occurs. If set to 'once', the sizing will occur only on the first navigation and the panel will remain at that size on subsequent navigations.

## **size\_panel\_to\_content\_axis**

Specifies the nature of the content sizing if the `size_panel_to_content` is not `false()`.

Use this property with care. Although the panel does its best to do what you expect, some decisions about content size and placement of scroll bars are determined by the browser using its own rules. This may lead to unexpected behaviour. In particular, layouts that use percentages to size elements may result in unexpected scroll bars.

### **Default value**

'both' – the height and width of the panel are resized.

### **Valid values**

Set to 'height' to size the panel's height only. Set to 'width' to size the panel's width only. Set to 'both' to size the panel's height and width.

## **scrolling**

Determines whether scrolling is permitted.

### **Default value**

'both'.

### **Valid values**

Set to 'yes' or 'no' to allow or disallow scrolling whether or not the content fits within the panel. Set to 'auto' to allow scrolling if required.

**class**

The Cascading Style Sheet class name of the weblet.

The class must be defined in an external CSS file which must be loaded into the current layout.

**Default value**

'std\_nav\_panel' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **transparent**

A Boolean property that sets the allowTransparent attribute of the IFRAME that makes up the panel. This is an Internet Explorer (7 and under) only attribute which, when true, allows the document loaded in the panel to have a transparent background-color.

It is important to note that this property does not make the panel transparent, it simply allows the panel contents to be transparent. The content document must still set its BODY background-color to transparent. If this property is false then any transparent background-color in the content document will be ignored.

This is a non-standard property that is only supported by Internet Explorer (7 and under) or Internet Explorer 8+ running in compatibility mode. Other browsers (including Internet Explorer 8+ running in standards mode) do not support this property. They will behave as if transparent is true. However, if no background color is specified in the content document, some browsers will default to transparent and other browsers will default to white. So, for best cross-browser results, leave this property as true and make sure you explicitly set the background color of your content document.

### **Default value**

true()

### **Valid values**

true() or false()

**nav\_url**

The URL the panel should navigate to.

**Default value**

Blank – the panel does not navigate to a URL.

**Valid values**

A valid URL, in single quotes. Note that a protocol (for example, http: or https:) must be specified.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

Blank.

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

**nav\_wamname**

The name of the WAM that will be invoked to display a page in the panel.

**Default value**

Blank – the current WAM will be invoked if the nav\_wname property is specified.

**Valid values**

The name of a WAM, in single quotes. A list of known WAM names may be chosen from by clicking the corresponding dropdown button in the property sheet.

**nav\_wrname**

The name of the Webroutine that will be invoked to display a page in the panel.

**Default value**

Blank – a Webroutine will not be invoked to display a page in the panel.

**Valid values**

The name of a Webroutine, in single quotes. A list of valid Webroutine names may be chosen from by clicking the corresponding dropdown button in the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **nav\_asynchronously**

A Boolean property that controls the way in which the panel is loaded if a URL has been specified in the nav\_url property. Ignored if nav\_url is not specified.

If true, the url is loaded using a new thread. The effect of this is that the browser will not wait for the panel to load before firing the onLoad event to indicate the page has loaded. If **nav\_asynchronously** is false, the browser will wait for the panel to finish loading before firing the event.

<p>This property is not supported by Opera. Opera will behave as if <b>nav_asynchronously</b> is true regardless of its actual value.</p>
---

### **Default value**

true() – the nav\_url URL (if specified) is navigated to asynchronously.

### **Valid values**

If set to True, the URL specified in the nav\_url property will be navigated to asynchronously (that is, the rest of the outer page will load before the panel is loaded).

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

Blank – a re-entry field is not used.

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

Blank – a re-entry field is not used.

**Valid values**

Any appropriate literal.

**wait\_content**

The text to be displayed whilst the panel is loading. HTML can also be specified.

**Default value**

Blank – no text is displayed whilst the panel is loading.

**Valid values**

Any appropriate literal, field name, system variable name or multilingual text variable name. A field, system variable or multilingual variable name can be chosen from a list by clicking the corresponding ellipses button in the property sheet.

## **wait\_content\_timeout**

If the wait\_content property is specified, this property specifies, in milliseconds, the amount of time to wait before the page is requested. This allows time for the image to load and be displayed.

### **Default value**

100 – if the wait\_content property is specified, a delay of 100 milliseconds occurs before the navigation occurs.

### **Valid values**

A valid wait time, in milliseconds.

**wait\_content\_class**

The Cascading Style Sheet class to be applied to the container element of the content specified in the wait\_content property.

**Default value**

Blank – no style is applied.

**Valid values**

The name of a valid Cascading Style Sheet class, in single quotes. A valid style can be chosen from a list by clicking the corresponding dropdown button in the property sheet.

## **wait\_content\_relative\_image**

The path and file name, relative to the images directory, of the image to be displayed whilst the panel is loading. The image will initially be retrieved from the server, but will subsequently be retrieved from the browser cache, if enabled.

### **Default value**

Blank – no image is shown.

### **Valid values**

A valid path and image file name, in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **wait\_content\_absolute\_image**

The path and file name of the image to be displayed whilst the panel is loading. The image will initially be retrieved from the server, but will subsequently be retrieved from the browser cache, if enabled.

### **Default value**

Blank – no image is shown.

### **Valid values**

A valid path and file name, in single quotes. Should not be specified if the `wait_content_relative_image` property is specified.

## **wait\_content\_image\_alignment**

The alignment of the image specified in the wait\_content\_relative\_image or wait\_content\_absolute\_image properties, relative to the wait\_content property.

### **Default value**

Blank – equivalent to 'left'.

### **Valid values**

Set to 'left', 'right', 'top' or 'bottom'.

## **wait\_content\_image\_class**

The Cascading Style Sheet class of the image specified in the wait\_content\_relative\_image or wait\_content\_absolute\_image property.

### **Default value**

Blank – the image has no class.

### **Valid values**

The name of a valid Cascading Style Sheet class, in single quotes. A valid style can be chosen from a list by clicking the corresponding dropdown button in the property sheet.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.22 Panel (std\_panel)

[QuickStart - Panel Properties - Panel](#)

The Panel weblet is used as a container for other controls on a web page. It is typically used to ensure that certain portions of a web page are positioned absolutely as opposed to relatively. As such, it is usually used in conjunction with other 'arrangement' controls such as tables and attachment panels.

## **QuickStart - Panel**

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Panel weblet.
2. Drag and drop it onto the web page being designed.
3. Size it appropriately by specifying the height and width properties or by dragging the panel's sizing handles. Other controls can now be dragged onto it. Note that these controls are positioned absolutely as opposed to relatively.

## Properties - Panel

The Panel weblet's properties are:

border          height   pos\_absolute  
border\_width   hide\_if   snap\_to\_grid  
class            name      width  
grid\_size        panes

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name in single quotes.

## **panes**

An XML nodeset specifying a set of panes to show. This is a system generated value set up when you drag the banner into a pane on the design view. Cannot be modified.

### **Default value**

`document(")/*/lxml:data/lxml:panes[@id='<unique id>']` (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

### **Valid values**

Not Applicable. (This value is system maintained.)

## **border**

The style of the border.

### **Default value**

Blank. The weblet has no border.

### **Valid values**

A valid border style, in single quotes. A valid style can be chosen from a list by clicking the corresponding dropdown button in the property sheet. Note that 'window-inset' is only supported by Internet Explorer.

**border\_width**

The width of the panel's border.

**Default value**

Blank. Equivalent to the panel having no border.

**Valid values**

A width, in a valid unit of measurement, in single quotes. Alternatively, a width of 'thick', 'medium' or 'thin' can be selected by clicking on the corresponding dropdown button in the property sheet.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_panel' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **snap\_to\_grid**

A Boolean property that determines whether the Snap to Grid feature should be enabled at design time. Snap to Grid allows controls placed on the panel to be aligned more easily. This is a design-time feature only.

### **Default value**

true() – the Snap to Grid feature is enabled for the panel at design time.

### **Valid values**

Set to true() to enable Snap to Grid or false() to disable it.

**grid\_size**

The distance between the points of the grid at design-time. The points allow easier alignment of controls placed on the grid. This is a design-time feature only.

**Default value**

'10px' – the points of the grid are placed 10 pixels apart.

**Valid values**

A distance, in a valid unit of measurement, in single quotes.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the panel on the web page.

**Default value**

'400pt'.

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the panel on the web page.

**Default value**

'200pt'.

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.1.23 Print Page (std\_printpage)

[QuickStart - Print Page Properties - Print Page](#)

The Print Page weblet provides a hyperlink to print the current page. It look like this:



## **QuickStart - Print Page**

To add the Print Page weblet to your web page:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Print Page weblets.
2. Drag and drop the weblet onto the web page.

## Properties - Print Page

The Print Page weblet properties are:

<a href="#">absolute_image_path</a>	<a href="#">disabled_class</a>	<a href="#">pos_absolute</a>
<a href="#">caption</a>	<a href="#">height_design</a>	<a href="#">relative_image_path</a>
<a href="#">class</a>	<a href="#">hide_focus</a>	<a href="#">tab_index</a>
<a href="#">disabled</a>	<a href="#">hide_if</a>	<a href="#">width_design</a>

**caption**

The caption for the weblet.

**Default value**

'Print'

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or any valid expression that returns True or False.

## **hide\_focus**

A Boolean property that, if evaluated to be True, will hide the focus rectangle for the weblet when it has focus.

### **Default value**

true()

### **Valid values**

true(), false() or a valid expression that returns a Boolean value.

**relative\_image\_path**

The path and file name, relative to the 'images' directory, of the image to be displayed. If specified, the absolute\_image\_path property should be left blank.

**Default value**

'icons/normal/16/printer\_16.png'

## **absolute\_image\_path**

The path and file name of the image to be displayed. If specified, the `relative_image_path` property should be left blank.

### **Default value**

The default is to use the image specified in the `relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_printpage' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **disabled\_class**

The Cascading Style Sheet of the weblet when the disabled property is set to True.

### **Default value**

'std\_printpage\_disabled' - The name of the shipped disabled class for the weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab index**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

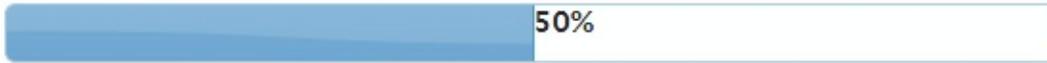
### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet

## 8.1.24 Progress bar (std\_progressbar)

### Properties – Progress bar

Display status of a determinate or indeterminate process. It can also be used to display a value as a percentage of its maximum value.



## Properties – Progress bar

caption	indeterminate	transitory
delayClose	max	value
height	name	width
hide_if	overlay	

**name**

The name of the weblet. If the weblet represents a field from your WAM, this should be the name of the field. If the weblet does not represent a field, you may wish to use your own name if using JavaScript or XSL that references the weblet. If nominating a name is not a consideration, the default name should be used, as determined by LANSA.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name, in single quotes. This can be a nominal choice, or the name of a field, the value of which is set by the weblet.

**value**

The value to set the weblet to. The ratio of this value to the maximum value is used to represent the current progress.

**Default value**

Zero: 0% of progress

**Valid values**

0 to maximum value.

**max**

The value that represents 100% of progress. This, together with the current value are used to determine the progress shown by the progress bar.

**Default value**

100

**Valid values**

A numeric value greater than zero.

**height**

The height of the progress bar.

**Default value**

Auto: 20px

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**width**

The width of the progress bar.

**Default value**

Auto: 99%

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**caption**

Caption to show on progress bar once it reaches its maximum value or when the indeterminate progress bar is shown.

**Default value**

Auto: "In progress ..." or "Complete"

**Valid values**

Single-quoted text.

**transitory**

Show progress bar only while its value is greater than zero until it reaches its maximum value.

**Default value**

False()

**Valid values**

Any valid XPath expression that returns a Boolean value.

**indeterminate**

Set to true to show a progress bar that indicates something is in progress, but progress is indeterminate.

**Default value**

False()

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **overlay**

For transitory progress bars: Overlay the progress bar at the center of the viewport. This prevents users from interacting with the page while the progress bar is displayed.

### **Default value**

False()

### **Valid values**

Any valid XPath expression that returns a Boolean value.

## **delayClose**

For transitory progress bars: Delay time (in milliseconds) to keep the progress bar before closing it.

### **Default value**

500

### **Valid values**

An integer value

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## 8.1.25 Prompter (std\_prompter)

### Properties - Prompter

The Prompter weblet provides a mechanism through which a user can select a value for a field, or fields, using a custom user interface presented in a pop-up panel.

The weblet creates a button like this:



When clicked, the prompter will call a webroutine and display its output in a panel like this:

Employee Number	<input type="text"/>		BROWN, VERONICA
Employee Surname	<input type="text"/>		BLOGGS, FRED JOHN ALAN
			SMITHSON, FRED
			MISS SIMPSON, ANNE
			JONES, BEN
			SMYTHE, JOHN
			SMITHE, Robert
			SMITHSON, PAUL
			SMITHS, PETER
			SMITHERS, JACK
			SNELL, GEORGE
			SNEDDON, ALLAN
			SNASHALL, DAMIAN
			PERRY WILLIAM

This webroutine may call other webroutines as necessary, for example, pages in a list, steps in a wizard. When the final value is known, it is output in a predefined webroutine known as the "closing" webroutine. The prompter will recognize the closing webroutine, copy the value(s) into the current page and close. See the [closing\\_wname](#), [field\\_mapping](#) and [field\\_name\\_to\\_exchange](#) properties for more details on closing the prompter and accessing its results.

## Properties - Prompter

The Prompter weblet's properties are:

<code>absolute_image_path</code>	<code>field_name_to_exchange</code>	<code>pre_show_js</code>
<code>auto_resize</code>	<code>formname</code>	<code>prompter_class</code>
<code>border</code>	<code>hide_if</code>	<code>prompter_height</code>
<code>border_width</code>	<code>image_height</code>	<code>prompter_url</code>
<code>button_class</code>	<code>image_width</code>	<code>prompter_wamname</code>
<code>button_height</code>	<code>name</code>	<code>prompter_width</code>
<code>button_mouseover_class</code>	<code>on_change_protocol</code>	<code>prompter_wname</code>
<code>button_width</code>	<code>on_change_reentryfield</code>	<code>protocol</code>
<code>caption</code>	<code>on_change_reentryvalue</code>	<code>reentryfield</code>
<code>closing_url</code>	<code>on_change_target_window_name</code>	<code>reentryvalue</code>
<code>closing_wname</code>	<code>on_change_wamname</code>	<code>relative_image_path</code>
<code>disabled</code>	<code>on_change_wname</code>	<code>tab_index</code>
<code>field_mapping</code>	<code>pos_absolute</code>	<code>title</code>
		<code>vf_wamevent</code>

**name**

The name of the weblet. If the weblet represents a field from your WAM, this should be the name of the field. If the weblet does not represent a field, you may wish to use your own name if using JavaScript or XSL that references the weblet. If nominating a name is not a consideration, the default name should be used, as determined by LANSA.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name, in single quotes. This can be a nominal choice, or the name of a field, the value of which is set by the weblet.

**caption**

The caption to appear to the right of the prompter's image.

**Default value**

Blank – the prompter has no text to the right of the image.

**Valid values**

Single-quoted text or the name of a field, multilingual text variable or system variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **relative\_image\_path**

The path and file name, relative to the 'images' directory, of the image to be displayed. If specified, the absolute\_image\_path property should be left blank.

### **Default value**

'search\_1.gif'

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **absolute\_image\_path**

The path and file name of the image to be displayed. If specified, the relative\_image\_path property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the relative\_image\_path property.

### **Valid values**

The path and name of an image enclosed in single quotes.

**image\_height**

The height of the image on the prompter button.

**Default value**

'12pt'.

**Valid values**

A height, in a valid unit of measure, in single quotes.

**image\_width**

The width of the image on the prompter button.

**Default value**

'12pt'.

**Valid values**

A width, in a valid unit of measure, in single quotes.

## **border**

The style of the prompter's border.

### **Default value**

Blank (this is equivalent to the weblet adopting its default style).

### **Valid values**

A fixed set of border style – 'dashed', 'dotted', 'double', 'groove', 'inset', 'outset', 'ridge', 'solid', 'window-inset'. Note that 'window-inset' is only supported by Internet Explorer.

**border\_width**

The width of the weblet's border.

**Default value**

Blank. Equivalent to the weblet adopting its default width.

**Valid values**

A width, in a valid unit of measurement, in single quotes.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

false() – the weblet will always be shown

### Valid values

True(), False() or any valid expression that returns True or False.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

'left:0pt;top:0pt;' – equivalent to the weblet being positioned relatively.

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**button\_width**

The width of the prompter button.

**Default value**

Blank. Equivalent to the button adopting its default width.

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**button\_height**

The height of the prompter button.

**Default value**

Blank. Equivalent to the button adopting its default height.

**Valid values**

A height, in a valid unit of measurement, in single quotes.

## **prompter\_width**

The initial width of the prompt window when the prompt button is clicked. If *auto\_resize* is set to true then the prompter will automatically resize when its content is loaded.

### **Default value**

Blank. Equivalent to the prompt window adopting a default width.

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **prompter\_height**

The initial height of the prompt window when the prompt button is clicked. If *auto\_resize* is set to true then the prompter will automatically resize when its content is loaded.

### **Default value**

Blank. Equivalent to the prompt window adopting its default height.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **auto\_resize**

Determines whether the prompter will automatically resize to the size of its content.

### **Default value**

true

### **Valid values**

true or false. When true, the prompter will automatically resize to match the size of the page displayed inside it. If false, the prompter will be sized according to the *prompter\_width* and *prompter\_height* properties.

The size used will be determined by the browser and may vary slightly from browser to browser. It may also be affected by the initial size set with the *prompter\_width* and *prompter\_height* properties. For best results, make sure your prompter content page explicitly sizes its content so that the browser doesn't have to decide for itself. Note that most browsers will expand the prompter if necessary but it will not make it smaller. Some browsers may make it smaller vertically but not horizontally.

**button\_class**

The Cascading Style Sheet class of the prompter button.

**Default value**

'std\_prompter\_button'. This is the default class for the prompter button and is provided with the all shipped cascading style sheets.

**Valid values**

Any valid class name selected from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **prompter\_class**

The Cascading Style Sheet class of the prompt window shown when the prompter button is clicked.

### **Default value**

'std\_prompter'. This is the default class for the prompt window and is provided with the all shipped cascading style sheets.

### **Valid values**

Any valid class name selected from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **button\_mouseover\_class**

The Cascading Style Sheet class of the prompter button when the mouse is moved over it.

### **Default value**

'std\_prompter\_button\_mouseover'. This is the default mouseover class for the prompter button and is provided with all shipped cascading style sheets.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **prompter\_url**

The URL to navigate the prompter panel to. Will be ignored if the prompter\_wname property is specified.

### **Default value**

Blank – the prompter does not navigate to a URL.

### **Valid values**

A valid URL, in single quotes. Note that a protocol (for example, http: or https:) must be specified.

The URL must be from the same server that the WAM is from. Browser security restrictions prevent code in a page from one server accessing information in a page from another server. Using a URL that points to another server will cause the prompter code to fail with a "permission denied" error.

**prompter\_wamname**

The name of the WAM to be invoked to display a page in the prompter window from which a selection can be made. Should be left blank if the prompter\_url property is specified.

**Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

**Valid values**

A valid WAM name, in single quotes. Click on the corresponding dropdown button in the property sheet to choose from a list of known WAMs.

**prompter\_wrname**

The name of the Webroutine to be invoked to display a page in the prompter window from which a selection can be made. Should be left blank if the prompter\_url property is specified.

**Default value**

A Webroutine name must be specified if the prompter\_wrname property is specified.

**Valid values**

A valid Webroutine name, in single quotes. Click on the corresponding dropdown button in the property sheet to choose from a list of known Webroutines.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **field\_name\_to\_exchange**

The name of the field in the local WEBROUTINE (the WEBROUTINE containing the prompter) to be filled from the result of the prompter. This value is taken from a field of the same name in the [closing WEBROUTINE](#). To copy more than one field, or from fields with different names, use the [field\\_mapping](#) property.

### **Default value**

Blank. A field name must be specified.

### **Valid values**

A valid field name, in single quotes. Click on the corresponding dropdown button in the property sheet to choose from a list of known fields.

## **closing\_url**

The URL that will close the prompter window. Will be ignored if the closing\_wname property is specified.

### **Default value**

Blank. A URL is not invoked to close the prompter window.

### **Valid values**

A valid URL, in single quotes. Note that a protocol (for example, http: or https:) must be specified.

The URL must be from the same server that the WAM is from. Browser security restrictions prevent code in a page from one server accessing information in a page from another server. Using a URL that points to another server will cause the prompter code to fail with a "permission denied" error.

## field\_mapping

Used to define the mapping of multiple fields in the [closing WEBROUTINE](#) into the local WEBROUTINE (the WEBROUTINE containing the prompter). This property is not directly editable. You can open a custom property editor window by clicking the ellipse button in the property sheet.

If you have only one field to copy from the closing WEBROUTINE you can use the [field\\_name\\_to\\_exchange](#) property instead. If you have one or more fields to copy, all with matching names (i.e. FIELDA maps to FIELDA, FIELDB maps to FIELDB, etc.) then you can leave *field\_mapping* and *field\_name\_to\_exchange* unchanged. The prompter will automatically copy all fields in the closing WEBROUTINE into matching fields (if present) in the local WEBROUTINE.

Note that if *field\_name\_to\_exchange* is defined *field\_mapping* will be ignored.

### Default value

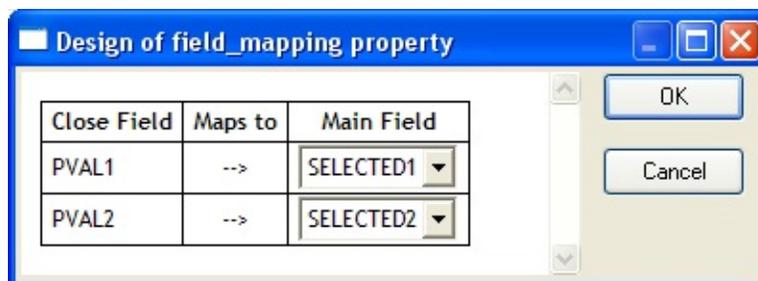
This [XPath](#) expression identifies an XML data fragment, stored in the XSL Source, that contains the mapping information:

```
document(")/*/lxml:data/lxml:prompter[@id='<unique id>']
```

### Valid values

Not Applicable. (This value is system maintained.) Use the ellipses button in the property sheet to edit field mapping.

In the following example the local WEBROUTINE is 'Main' and the prompter closing WEBROUTINE is 'Close'. The fields #PVAL1 and #PVAL2 from *Close* will be copied into the fields #SELECTED1 and #SELECTED2 in *Main*.



## **closing\_wrname**

The name of the Webroutine, in the current WAM, to be invoked that will close the prompter panel. Leave blank if the closing\_url property is specified.

The prompter will close as soon as this webroutine is loaded, however the webroutine design may flash briefly in the user interface so you should use a blank layout with no visible content.

All of the output values for the prompter must exist in the HTML output of the webroutine as <input> elements for the Prompter to be able to find them and transfer their values. The quickest way to do this and keep them invisible is to define them as \*HIDDEN in your web map before generating the HTML.

## **Default value**

'Close' – this is a suggested Webroutine name only.

## **Valid values**

The name of a valid Webroutine in the current WAM. Click on the corresponding dropdown button in the property sheet to select from a list of known Webroutines.

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

'STDREENTRY'

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

True(), false() or a valid expression.

**title**

Tool Tip text.

**Default value**

Blank – no Tool Tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **on\_change\_wamname**

The name of the WAM to be invoked after a selection has been made in the prompter. This property would typically be used if the value returned by the prompter is to be directed to a WAM other than the current one.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_change\_wrname**

The name of the Webroutine to be invoked after a selection has been made in the prompter.

**Default value**

Blank – no Webroutine is invoked when a selection is made in the prompter.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_change\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_change\_protocol**

The protocol (for example, http: or https:) the weblet should use for navigation to the WAM specified in the on\_change\_wamname property after a selection has been made in the prompter.

**Default value**

Blank – the current protocol is used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **on\_change\_reentryfield**

A field name to be used to post the value specified in the on\_change\_reentryvalue property after a selection is made in the prompter. This field is only submitted if the on\_change\_wname property is specified.

### **Default value**

Blank – no field is used to post a value.

### **Valid values**

A valid field name, in single quotes. Click on the corresponding dropdown button in the property sheet to select from a list of know fields.

## **on\_change\_reentryvalue**

The value to be posted into the field specified on the on\_change\_reentryfield property. This value is only submitted if the on\_change\_wname property is specified.

### **Default value**

Blank – no value is posted.

### **Valid values**

A valid value for the type of field specified in the on\_change\_reentryfield property.

## **on\_change\_target\_window\_name**

The name of the window to which content is to be directed when the Webroutine specified in the on\_change\_wname property is invoked.

### **Default value**

Blank – content is directed to the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

**pre\_show\_js**

JavaScript code to be run prior to the prompter being shown.

**Default value**

Blank. No JavaScript is run.

**Valid values**

Any valid JavaScript function, or JavaScript code followed by a semicolon (;) or double backslash (//). A section of JavaScript followed by a semicolon indicates the script should flow into the existing submit JavaScript, whereas the double backslash indicates that the rest of the submit JavaScript for WAM submits is to be commented out.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.26 Radio Button (std\_rad\_button)

### [QuickStart - Radio Group Properties - Radio Group](#)

The radio button weblet displays a single radio button. It broadly corresponds to the `<input type="radio">` HTML element.

Radio buttons are typically grouped as two or more radio buttons for mutually exclusive selection. In other words, you can choose only one of several options—like a multiple-choice question. It looks like this:



but is typically grouped to look more like this:



You may choose to use this weblet instead of the `std_radbuttons` weblet which is implemented as a group of radio buttons, if you want to store the resulting selections in different fields.

The radio button weblet includes properties such as `on_click_wname` that allow it to navigate to another webroutine when clicked. It is not good user-interface design to initiate actions on the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

## QuickStart - Radio Button

A radio button should always be displayed as a group of two or more radio buttons. Each radio button you add to the web page will usually correspond to a field included on WEB\_MAP of a webroutine. When you open the XSL generated for the webroutine in the LANSA Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Radio Button weblet.
2. Drag the Radio Button weblet over the field generated from your WEB\_MAP definition. (You may want to delete the field description). This will display with default radio button settings.

 Caption

3. Click on the weblet to review the Details tab. Notice that the name and value properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be compared against the code property to determine if this radio button should be checked. When the radio buttons value is changed the appropriate value will be place in the field nominated on the name property – in this case the same field.
4. Set the caption property as required and ensure the code property matches the required value for the caption.

## Properties - Radio Button

The Radio Button weblet's properties are:

alignment	label_id	reentryfield
caption	mouseover_class	reentryvalue
class	name	tab_index
code	on_click_wamname	target_window_name
disabled	on_click_wname	text_class
formname	pos_absolute	value
hide_if	protocol	vf_wamevent

**name**

The name of the radio button. Normally, you would leave this as the default and let LANSa use its own internal naming convention. If the weblet has been dropped onto a field, or is to be used to display or populate a field, the field name is used.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field or a default value.

## Default value

Blank.

## Valid values

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

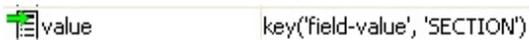
## Example

This example indicates the value should be set to the current value of the field #SECTION. When entered into the property this looks like this:



A screenshot of a property sheet. On the left, there is a small icon of a document with a green checkmark, followed by the text 'value'. To the right of this is a blue rectangular highlight. Further right, there is a text input field containing the text '#SECTION'.

When focus is moved off the property the same value will appear as follows:



A screenshot of a property sheet. On the left, there is a small icon of a document with a green checkmark, followed by the text 'value'. To the right of this is a text input field containing the text 'key('field-value', 'SECTION')'.

**caption**

The text to display next to the radio button.

**Default value**

Blank.

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

**code**

The value to be placed in the field when the radio button is checked.

**Default value**

Blank.

**Valid values**

Single-quoted text.

**label\_id**

An identifier for the radio button element which allows the radio button caption to be bound to the button, so that if the caption is clicked it will respond in the same way as if the button itself was clicked. This value must be unique.

**Default value**

concat(\$name,\$code) (that is, concatenation of the value is the name and code properties).

**Valid values**

Single-quoted text.

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

### **Default value**

'STDREENTRY'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## hide\_if

A Boolean property. An expression which, if evaluated to be True, will hide the weblet.

### Default value

false() (that is, the grid will always be shown)

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the grid if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

hide_if	#STD_FLAG = 'X'
---------	-----------------

When the property loses focus, the expression is shown as follows:

hide_if	key('field-value', 'STD_FLAG') = 'X'
---------	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **on\_click\_wamname**

The name of the WAM to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

The name of the Webroutine to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

Blank.

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_waname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown. A unique name can be entered or use the available selection for a predefined set of values.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

**alignment**

Indicates whether the caption should be displayed to the left or right of the radio buttons.

**Default value**

'right'

**Valid values**

'left' or 'right'

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognized. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**class**

The Cascading Style Sheet class to be applied to the weblet.

**Default value**

The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button' is supplied.

**mouseover\_class**

The Cascading Style Sheet class to be applied to the weblet when the mouse is moved over it.

**Default value**

Blank.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button\_mouseover' is supplied.

**text\_class**

The Cascading Style Sheet class to be applied to radio button captions in the weblet.

**Default value**

The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button\_text' is supplied.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

## 8.1.27 Radio Group (std\_radbuttons)

### QuickStart - Radio Button Properties - Radio Button

The radio group weblet builds groups of radio buttons. The number of buttons and button values can be established from a working list or a static set of values defined via the item property of the weblet. Each radio buttons broadly corresponds to an `<input type="radio">` HTML element. It looks like this:



Select a Color

- Red
- Yellow
- Green
- Blue

The radio buttons group weblet includes properties such as `on_click_wname` that allow it to navigate to another webroutine when clicked. It is not good user-interface design to initiate actions on the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

## QuickStart - Radio Group

Each entry in a radio button group is defined by an entry in a working list or a set of items hard coded in the radio group properties.

### If you use a working list:

To use a working list to define the radio buttons in the group, you need to create a webroutine that specifies a field to store the selected value and the working list of options in the WEB\_MAP. When you open the XSL generated for the webroutine in the LANSA Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Radio Group weblet.
2. Drag the Radio Group weblet onto the field to store the value and release the left-mouse button. (You may want to delete the field description). This will display with default radio button settings.

Radio 1  Radio 2

3. Click on the weblet to review the Details tab. Notice that the name and value properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected radio button. When the radio buttons value is changed the appropriate value will be placed in the field nominated on the name property – in this case the same field.

Change the listname property to the working list passed on the WEB\_MAP. The radio group representation should immediately change to a set of buttons without identifiers.

Set the codefield and captionfield properties to the appropriate fields from the working list.

### If you use the items property:

To use a set of items hard coded in the radio group properties, you would need to create a webroutine that specifies a field in its WEB\_MAP. When you open the XSL generated for the webroutine in the LANSA Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Radio Group weblet.
2. Drag the Radio Group weblet onto the field to store the value and release the left-mouse button. (You may want to delete the field description). This will

display with default radio button settings.

Radio 1  Radio 2

3. Click on the weblet to review the Details tab. Notice that the name and value properties have been set to indicate the field you dragged the weblet on to. The value property indicates that on presentation of the web page any value currently in this field will be used to set the selected radio button. When the radio buttons value is changed the appropriate value will be place in the field nominated on the name property – in this case the same field.
4. Set up the list of items to be used as radio buttons by selecting the ellipses button on the items property. Proceed to define the require entries for the radio buttons.

## Properties - Radio Group

The Radio Group weblet's properties are:

alignment	hide_if	reentryfield
captionfield	items	reentryvalue
class	listname	selector_field
codefield	mouseover_class	selector_value_eq
disabled	name	show_groupbox
display_mode	on_click_wamname	tab_index
formname	on_click_wrname	target_window_name
group_title	orientation	text_class
groupbox_class	pos_absolute	width
height	protocol	value
		vf_wamevent

**name**

The name of the radio group. Normally, you would leave this as the default and let LANSa use its own internal naming convention. If the weblet has been dropped onto a field, or is to be used to display or populate a field, the field name is used.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field or a default value.

## Default value

Blank.

## Valid values

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## Example

This example indicates the value should be set to the current value of the field #SECTION. When entered into the property this looks like this:



A screenshot of a property sheet showing a value field. The field contains the text "value" and is highlighted in blue. To its right is a field reference field containing the text "#SECTION".

When focus is moved off the property the same value will appear as follows:



A screenshot of a property sheet showing a value field. The field contains the text "value" and is highlighted in blue. To its right is a field reference field containing the text "key('field-value', 'SECTION')".

**display\_mode**

Controls whether the weblet accepts input or displays output.

**Default value**

'input'

**Valid values**

'input' or 'output'.

## items

An XML nodeset specifying the items to appear in the weblet. Can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. Leave blank if items are populated from a list specified in the listname property.

## Default value

document(")\*/lxml:data/lxml:dropdown (this indicates no items have been defined for this dropdown.)

## Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

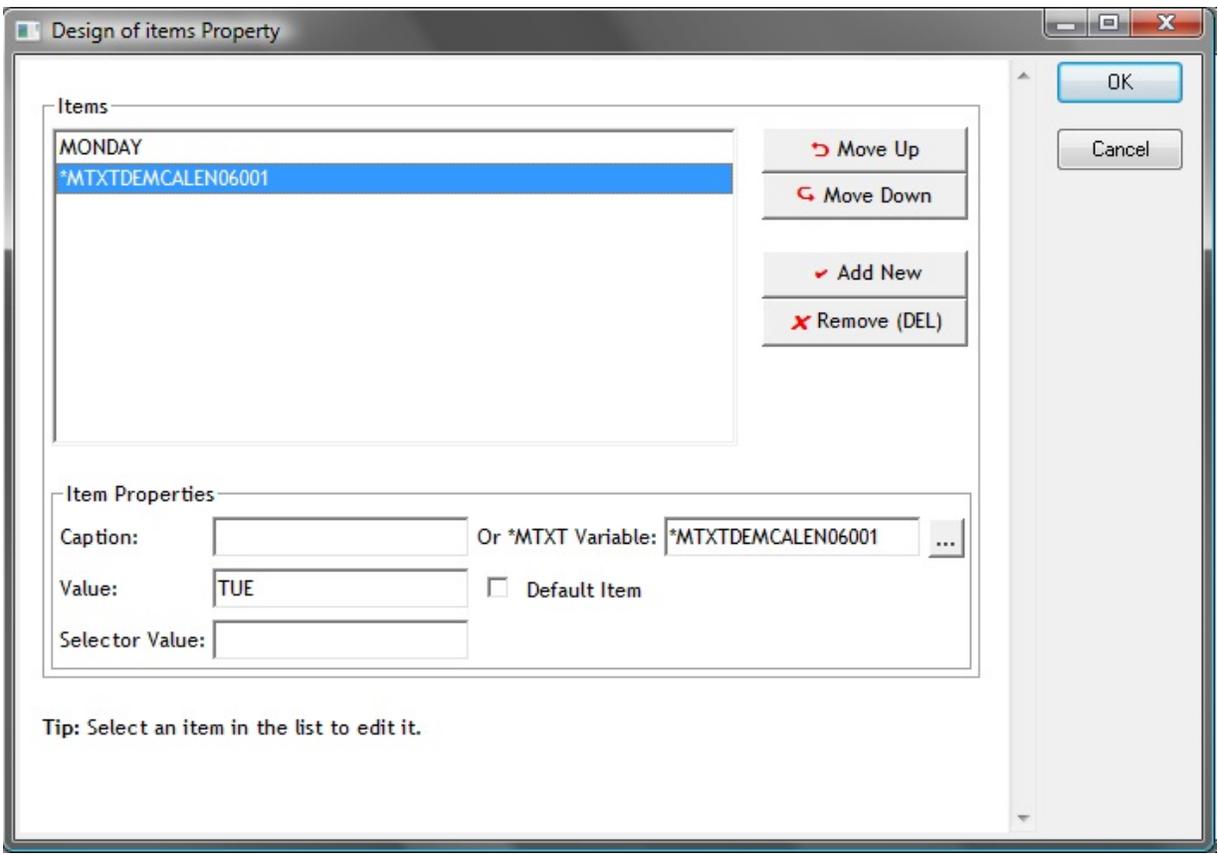
## Example

This example indicates that items have been setup in the designer to use as dropdown values.



The screenshot shows a property sheet with a tab labeled 'items'. The value of the property is `document(")*/lxml:data/lxml:caption_value_pairs[@id='4A14173FA6D64A059507F9124A630CF3']`.

Using the ellipse button on the property you will see the designer and be able to maintain the items to be displayed as radio buttons. The following view of the designer indicates two radio buttons are required in the radio group. The first entry has the literal value 'MONDAY' and the second entry uses a multilingual variable to display the description for the code TUE. Check the Default Item check box for the item which is to be selected if no value is preselected.



**listname**

The name of the working list to use to create the radio buttons in the radio group. Leave blank if details are specified in the items property.

**Default value**

Blank.

**Valid values**

Single-quoted text. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

**selector\_field**

The name of the field in the list specified in the listname property that can contain a value to limit, to a subset, the list items shown in the weblet. This property is used in conjunction with the selector\_value\_eq property.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

## **selector\_value\_eq**

This value is used in order to limit, to a subset, the list items shown in the weblet. If a listname property is provided the associated field must be specified in the selector\_field property. If the items property designer has been used to define the list of values, the corresponding selector value entered in the designer is used.

### **Default value**

Blank.

### **Valid values**

Single-quoted text or a numeric value. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**codefield**

The name of the field in the list specified in the listname property that holds the key value for each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**captionfield**

The name of the field in the list specified in the listname property that holds the caption for the each list item.

**Default value**

Blank.

**Valid values**

Single-quoted text. A field, from the working list nominated in listname, can be selected by clicking the corresponding dropdown button in the property sheet.

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

'STDREENTRY'

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## hide\_if

A Boolean property. An expression which, if evaluated to be True, will hide the weblet.

### Default value

false() (that is, the grid will always be shown)

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the grid if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

hide_if	#STD_FLAG = 'X'
---------	-----------------

When the property loses focus, the expression is shown as follows:

hide_if	key('field-value', 'STD_FLAG') = 'X'
---------	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **on\_click\_wamname**

The name of the WAM to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

The name of the Webroutine to be invoked when an item in the weblet is selected.

**Note:** It is not good user-interface design to initiate actions from the click of a radio button. Devices such as a push button, menu item or anchor (hyperlink) should be used to accomplish this.

## **Default value**

Blank.

## **Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_waname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine invoked by this weblet.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually 'http:' or 'https:'.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown. A unique name can be entered or use the available selection for a predefined set of values.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that \_search and \_media are supported by Internet Explorer 6 only.

## **alignment**

Indicates whether the caption should be displayed to the left or right of the radio buttons.

**Warning:** Using left alignment with a vertical orientation may cause your radio buttons to be misaligned as the justification is based on the first character of the caption for the radio buttons.

### **Default value**

'right'

### **Valid values**

'left' or 'right'

**orientation**

Indicates the direction in which the group of radio buttons should be oriented.

**Default value**

'horizontal'

**Valid values**

'vertical' or 'horizontal'

## **show\_groupbox**

A Boolean property, the result of which determines whether a group box will appear around the group of radio buttons.

### **Default value**

false() (that is, no group box will be shown).

### **Valid values**

true(), false() or a valid expression.

## **group\_title**

A title to be displayed on the group box. If the show\_groupbox property is set the title will appear at the top left of the group box. If the group box is not shown the title will appear centered above the group of radio buttons.

## **Default value**

Blank

## **Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognized. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**class**

The Cascading Style Sheet class to be applied to the weblet.

**Default value**

The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button' is supplied.

**mouseover\_class**

The Cascading Style Sheet class to be applied to the weblet when the mouse is moved over it.

**Default value**

Blank.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button\_mouseover' is supplied.

**text\_class**

The Cascading Style Sheet class to be applied to radio button captions in the weblet.

**Default value**

The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet. A shipped class of 'std\_rad\_button\_text' is supplied.

## **groupbox\_class**

The Cascading Style Sheet class to be applied to the group box associated with the weblet. This property is only applied if the `show_groupbox` property is set as true.

### **Default value**

Blank.

### **Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

### **Example**

```
groupbox_class 'std_listbox'
```

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

String value. Comma (',') not allowed.

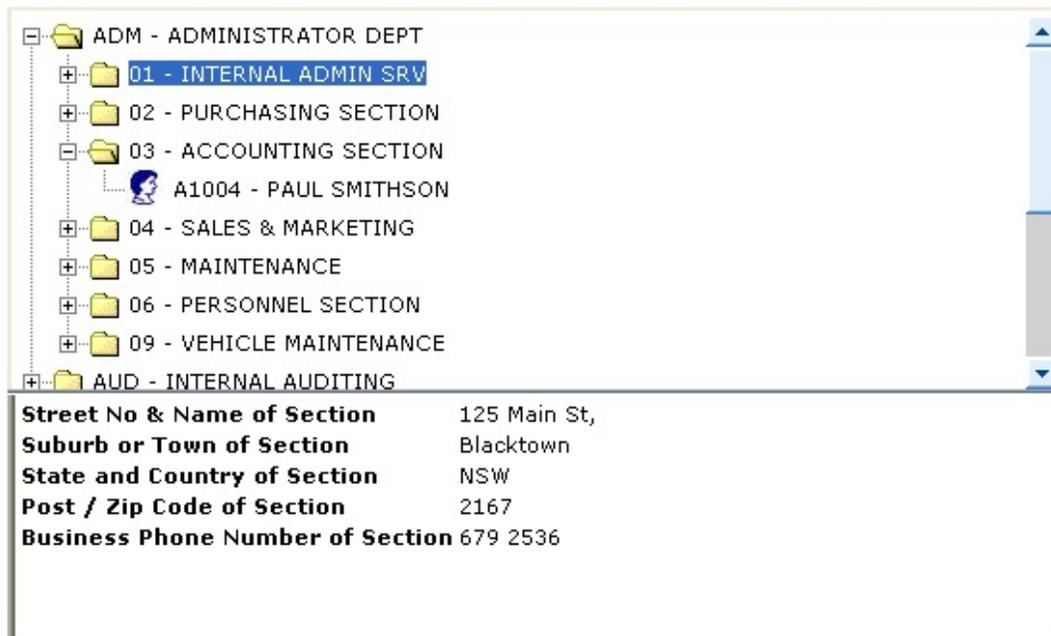
## 8.1.28 Horizontal Splitter (std\_splitter\_horz)

### [QuickStart - Horizontal Splitter Properties - Horizontal Splitter](#)

The Horizontal splitter weblet divides the area covered by the weblet into a top and bottom pane with a moveable divider separating the two panes. It does not provide any input or output but is used solely to assist with web page design. Other elements or weblets can be dropped into each pane, effectively creating multiple editable areas on the web page.

The horizontal splitter can be combined with additional horizontal splitters or vertical splitters to construct more complex pages.

It looks like this:



## **QuickStart - Horizontal Splitter**

To use the horizontal splitter create a webroutine. Any WEB\_MAPs are optional as they do not have a direct bearing on the splitter weblet. Add a horizontal splitter to the resulting web page by completing the following steps in the LANSa Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Horizontal Splitter weblet.
2. Drag and drop the weblet to the required location in the Design view. Make sure the weblet is selected and then click on the Details tab.
3. Set the top\_portion\_percent property as required. The appropriate value to use in this property may become more apparent after the content of the splitter panes has been defined.
4. Format the content of the splitter panes as required by adding weblets and / or field information into the top and bottom area of the splitter.

## Properties - Horizontal Splitter

The Horizontal Splitter weblet's properties are:

bottom\_border name top\_class  
bottom\_class panes top\_proportion\_percent  
bottom\_style pos\_absolute top\_style  
divider\_class top\_border width  
height

**name**

The name of the horizontal splitter. Normally, you would leave this as the default and let LANSAs use its own internal naming convention.

**Default value**

`concat('o', position(), '_LANSAs_n')` – this is the internal name given to the weblet by LANSAs.

**Valid values**

A name, in single quotes.

## **panes**

An XML nodeset specifying a set of panes to show. This is a system generated value set up when you drag the splitter onto the design view.

**Note:** This value cannot be modified and is for information only.

### **Default value**

document(")/\*/lxml:data/lxml:splitter\_panes[@id='<unique id>'] (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

### **Valid values**

Not Applicable. (This value is system maintained.)

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's bottom-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

'top:0pt;top:0pt;' (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'top' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

'100%' (this is equivalent to the weblet the full width available).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

### **Default value**

'200pt' when placed directly on the form element or '100%' if contained within another weblet.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

**top\_proportion\_percent**

The percentage of space initially occupied by the top pane of the horizontal splitter.

**Default value**

70

**Valid values**

A valid percentage, entered as a number (that is, without quotes).

## **top\_border**

The border style of the top pane of the splitter. This will apply to the top, left and right border of the top pane.

### **Default value**

'outset'.

### **Valid values**

A fixed set of border style – 'dashed', 'dotted', 'double', 'groove', 'inset', 'outset', 'ridge', 'solid', 'window-inset'. Note that 'window-inset' is only supported by Internet Explorer.

## **bottom\_border**

The border style of the bottom pane of the splitter. This will apply to the bottom, left and right border of the bottom pane.

### **Default value**

'inset'.

### **Valid values**

A fixed set of border style – 'dashed', 'dotted', 'double', 'groove', 'inset', 'outset', 'ridge', 'solid', 'window-inset'. Note that 'window-inset' is only supported by Internet Explorer.

**top\_class**

The Cascading Style Sheet class to be applied to the top pane of the splitter.

**Default value**

'std\_splitter\_horz\_top' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**divider\_class**

The Cascading Style Sheet class to be applied to the divider bar.

**Default value**

'std\_splitter\_vert\_divider' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**bottom\_class**

The Cascading Style Sheet class to be applied to the bottom pane of the splitter.

**Default value**

'std\_splitter\_vert\_bottom' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**top\_style**

The inline Cascading Style Sheet style of the top pane of the splitter. This will override any corresponding characteristics detailed in the Cascading Style Sheet class nominated in the top\_class property.

**Default value**

Blank.

**Valid values**

Single quoted text string comprised of one or more style declarations, where a declaration is made up of a property, a colon, and one or more values.

**bottom\_style**

The inline style to be applied of the bottom pane of the splitter. This will override any corresponding characteristics detailed in the Cascading Style Sheet class nominated in the bottom\_class property.

**Default value**

Blank.

**Valid values**

Single quoted text string comprised of one or more style declarations, where a declaration is made up of a property, a colon, and one or more values.

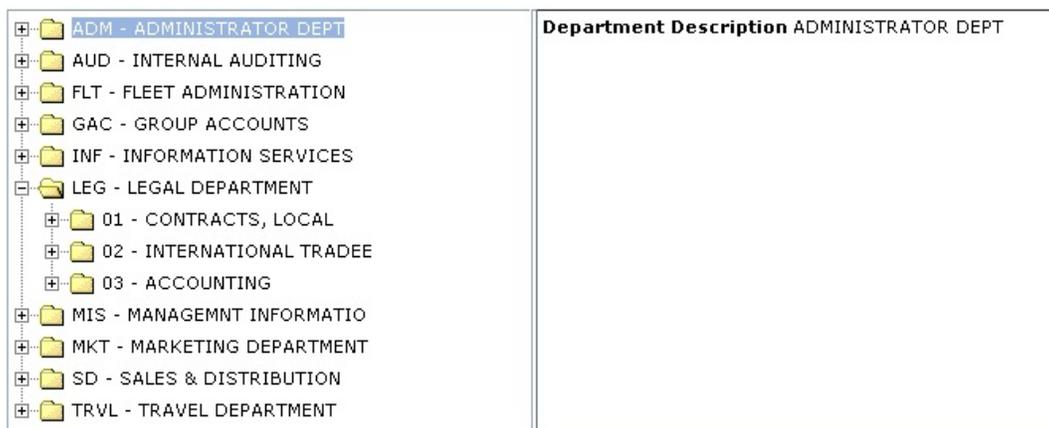
## 8.1.29 Vertical Splitter (std\_splitter\_vert)

### [QuickStart - Vertical Splitter Properties - Vertical Splitter](#)

The Vertical splitter weblet divides the area covered by the weblet into a left and right pane with a moveable divider separating the two panes. It does not provide any input or output but is used solely to assist with web page design. Other elements or weblets can be dropped into each pane, effectively creating multiple editable areas on the web page.

The vertical splitter can be combined with additional vertical splitters or horizontal splitters to construct more complex pages.

It looks like this:



## QuickStart - Vertical Splitter

To use the vertical splitter create a webroutine. Any WEB\_MAPs are optional as they do not have a direct bearing on the splitter weblet. Add a vertical splitter to the resulting web page by completing the following steps in the LANSA Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Vertical Splitter weblet.
2. Drag and drop the weblet to the required location in the Design view. Make sure the weblet is selected and then click on the Details tab.
3. Set the `left_portion_percent` property as required. The appropriate value to use in this property may become more apparent after the content of the splitter panes has been defined.
4. Format the content of the splitter panes as required by adding weblets and / or field information into the left and right area of the splitter.

## Properties - Vertical Splitter

The Vertical Splitter weblet's properties are:

divider_class	left_style	right_border
height	name	right_class
left_border	panes	right_style
left_class	pos_absolute	width
left_proportion_percent		

**name**

The name of the vertical splitter. Normally, you would leave this as the default and let LANSa use its own internal naming convention.

**Default value**

`concat('o', position(), '_LANSa_n')` – this is the internal name given to the weblet by LANSa.

**Valid values**

A name, in single quotes.

## **panes**

An XML nodeset specifying a set of panes to show. This is a system generated value set up when you drag the splitter onto the design view.

**Note:** This value cannot be modified and is for information only.

### **Default value**

document(")/\*//lxml:data/lxml:splitter\_panes[@id='<unique id>'] (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

### **Valid values**

Not Applicable. (This value is system maintained.)

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

'100%' (this is equivalent to the weblet the full width available).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

### **Default value**

'200pt' when placed directly on the form element or '100%' if contained within another weblet.

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **left\_proportion\_percent**

The percentage of space initially occupied by the left pane of the vertical splitter.

### **Default value**

30

### **Valid values**

A valid percentage, entered as a number (that is, without quotes).

## **left\_border**

The border style of the left pane of the splitter. This will apply to the top, bottom and left border of the left pane.

### **Default value**

'outset'.

### **Valid values**

A fixed set of border style – 'dashed', 'dotted', 'double', 'groove', 'inset', 'outset', 'ridge', 'solid', 'window-inset'. Note that 'window-inset' is only supported by Internet Explorer.

## **right\_border**

The border style of the right pane of the splitter. This will apply to the top, bottom and right border of the right pane.

### **Default value**

'inset'.

### **Valid values**

A fixed set of border style – 'dashed', 'dotted', 'double', 'groove', 'inset', 'outset', 'ridge', 'solid', 'window-inset'. Note that 'window-inset' is only supported by Internet Explorer.

**left\_class**

The Cascading Style Sheet class to be applied to the left pane of the splitter.

**Default value**

'std\_splitter\_vert\_left' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**divider\_class**

The Cascading Style Sheet class to be applied to the divider bar.

**Default value**

'std\_splitter\_vert\_divider' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**right\_class**

The Cascading Style Sheet class to be applied to the right pane of the splitter.

**Default value**

'std\_splitter\_vert\_right' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **left\_style**

The inline Cascading Style Sheet style of the left pane of the splitter. This will override any corresponding characteristics detailed in the Cascading Style Sheet class nominated in the left\_class property.

### **Default value**

Blank.

### **Valid values**

Single quoted text string comprised of one or more style declarations, where a declaration is made up of a property, a colon, and one or more values.

**right\_style**

The inline style to be applied of the right pane of the splitter. This will override any corresponding characteristics detailed in the Cascading Style Sheet class nominated in the right\_class property.

**Default value**

Blank.

**Valid values**

Single quoted text string comprised of one or more style declarations, where a declaration is made up of a property, a colon, and one or more values.

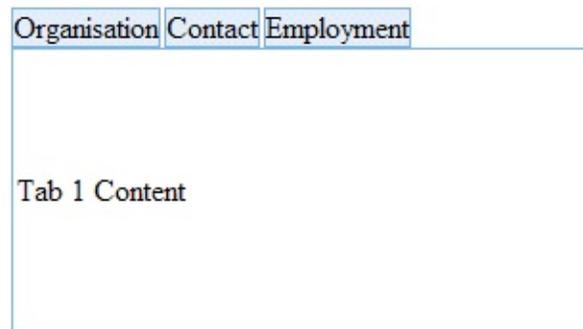
### 8.1.30 Tab Pages (std\_tab\_pages\_v2)

[QuickStart - Tab pages](#)

[Properties - Tab pages](#)

[Using the Tab Item Designer](#) [Using CSS with the Tab Pages weblet](#)

The Tab pages weblet provides the functionality of a tab control that can be used to organize related information into groups. This is what the Tab pages weblet looks like when embedded in a web page. In this example, common information is grouped at the top of the page. The tab control appears underneath that and provides access to three groups of information about an employee – Organization, Contacts and Employment. The Contacts tab is shown but the user can display information in the other groups by clicking the tabs.



The weblet provides an extensive set of properties that affect its appearance and behavior. The tab items themselves are typically specified statically using the tab item designer. In some cases you may wish to override properties of the tab items dynamically and you can do this by specifying the listname and related properties.

You can add content directly to each tab page if the containing webroutine provides the content in its web\_map. You do this in the usual ways – for example by dragging and dropping fields from the fields tab. Similarly you can add other weblets to the tab pages.

Alternatively you can add a navigation panel (std\_nav\_panel) weblet to one or more of your tab pages and set properties for each navigation panel so that it displays content that is provided by another webroutine (or URL).

## QuickStart - Tab pages

To use the Tab pages weblet, open your webroutine in the LANSA Editor and follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Tab pages weblet.
2. Drag and drop the weblet onto your page in the Design view Make sure the weblet on the page is selected and then click on the Details tab.
3. Click the ellipsis button next to the tabs property to open the tab items designer and define your tabs Click OK to close the tab items designer when complete.
4. In the Design view, click on your first tab to activate its page.
5. Add content to the tab page by cutting and pasting content from elsewhere on your web page or by dragging and dropping fields and/or weblets onto the tab page If you will be adding several fields you may wish to begin by adding a table to contain them – to do so, right-click in the tab page and select Insert, Table from the pop-up menu.
6. If you prefer that your tab page displays content provided by another webroutine, then you need to add a navigation panel and set its properties To do so follow these steps:
  - a. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Navigation panel weblet.
  - b. Drag and drop the weblet onto your tab page in the Design view Make sure the Navigation panel weblet on the tab page is selected and then click on the Details tab.
  - c. Set the nav\_wamname and nav\_wname properties (or the nav\_url property) to identify the webroutine (or url) that provides the content You may wish to set other properties for the Navigation panel such as the wait\_content and related properties.
7. Repeat from step 4 for each tab that you have defined in your tab pages weblet.

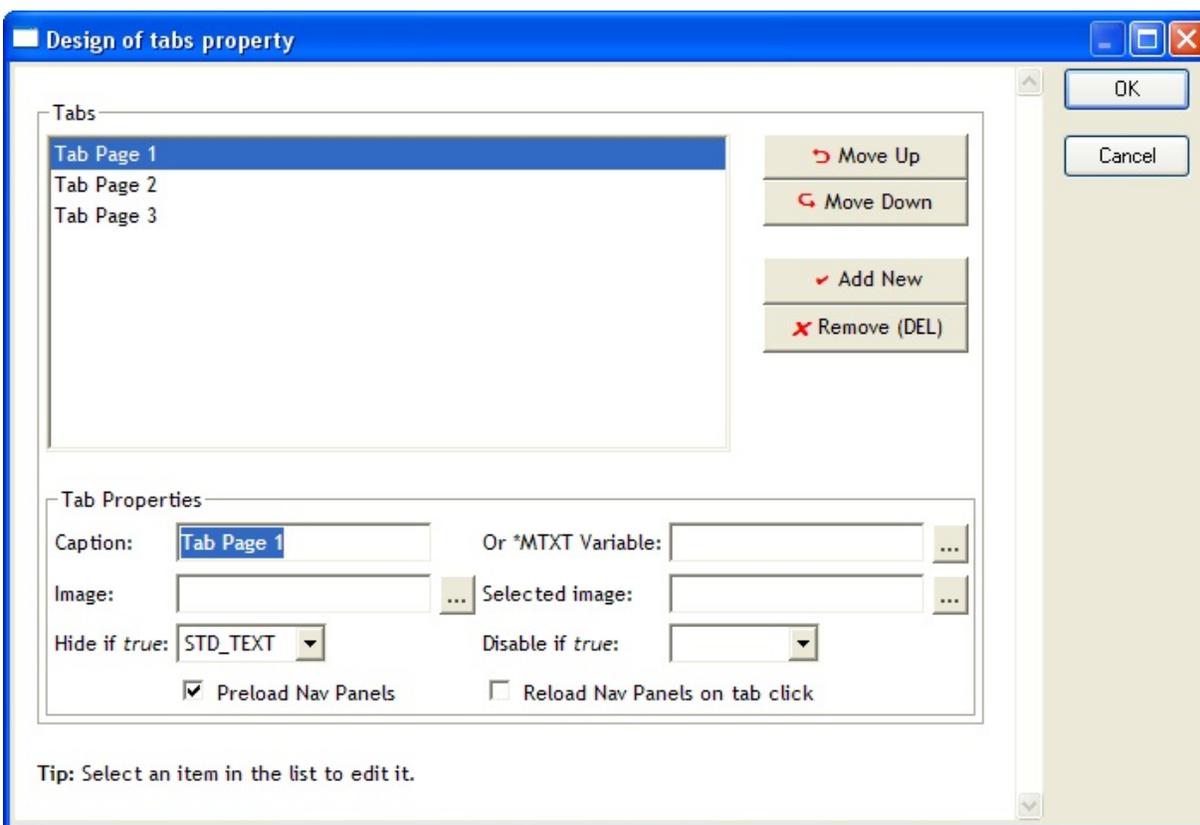
Note: A bug in current versions of Internet Explorer causes the border to disappear from around the content area of empty tabs. To prevent this behaviour in Internet Explorer, make sure all your tabs contain

something.

## Using the Tab Item Designer

To open the tab item designer, follow these steps:

1. Make sure the Tab pages weblet on your page is selected and then click on the Details tab.
2. Move the mouse pointer over the tabs property in the Details tab An ellipsis should appear next to the property value. Click the ellipsis button to open the tab items designer A window like the one shown below appears:



The window shows a list of the tab items presently defined for the weblet. You can click on an item to:

- change its properties;
- move the item up or down or remove it by using the buttons.

In *Tab Properties* area you can specify the properties for the selected item as follows:

### Tab Caption

Specifies the text that appears on the tab item If you are creating multilingual applications you should use the \*MTXT Variable instead. You can only specify

one of the Tab Caption or the \*MTXT Variable.

### **Or \*MTXT Variable**

Specifies the name of a multilingual text variable that contains the text that appears on the tab item. Click the ellipsis next to this property to select from a list of multilingual text variables. You can only specify one of the Tab Caption or the \*MTXT Variable.

### **Image**

Specifies the path and file name, relative to the images virtual directory, of an image to be displayed on the tab. If specified, this overrides any image specified in the tab\_image property.

### **Selected Image**

Specifies the path and file name, relative to the image's virtual directory, of an image to be displayed on the tab when it is selected. If specified, this overrides any image specified in the tab\_selected\_image property.

### **Hide if true**

Specifies the name of a field that indicates if the tab item should be hidden. Hidden, in this context, means that the tab and it's content is not sent to the browser at all. The tab is hidden if the field contains the boolean value **true**, the number **1** or a string value of **true**, **TRUE**, **y**, **Y** or **1**.

### **Disable if true**

Specifies the name of a field that indicates if the tab item should be disabled. The tab is disabled if the field contains the boolean value **true**, the number **1** or a string value of **true**, **TRUE**, **y**, **Y** or **1**.

### **Preload Nav Panels**

If the tab page contains a navigation panel, this checkbox controls whether the navigation panel for unselected tabs is loaded when the page containing the tab pages is loaded. If the webroutine or URL that is accessed by the navigation panel is resource-expensive or simply if you have a lot of tabs, you may wish to uncheck this box for one or all tab pages. If this checkbox is unchecked, the navigation panel is not loaded unless or until the corresponding tab page is selected.

### **Reload Nav Panels on tab click**

If the tab page contains a navigation panel, this checkbox controls whether the navigation panel is loaded once (when first displayed) or every time that the tab page is displayed.

## Using CSS with the Tab Pages weblet

### How the Default CSS Works Adding your own CSS Styles

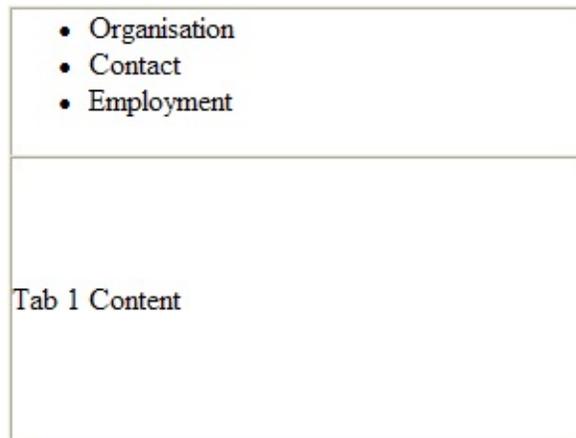
The Tab Pages weblet is constructed using an HTML table. The tabs themselves are constructed using an unordered list. The HTML for a Tab Pages weblet looks something like this:

```
<table id="TabPagesName" class="std_tab_pages" cellpadding="0" cellspacing="0">
  <tbody>
    <tr>
      <td class="std_tab_pages_top_tabs">
        <ul class="std_tab_pages_tabs">
          <li class="std_tab_active">
            <a onclick="return stdTabPagesTabClicked(this, 1)">Organisation</a>
          </li>
          <li>
            <a onclick="return stdTabPagesTabClicked(this, 2)">Contact</a>
          </li>
          <li>
            <a onclick="return stdTabPagesTabClicked(this, 3)">Employment</a>
          </li>
        </ul>
      </td>
    </tr>
    <tr>
      <td class="std_tab_pages_content_wrapper">
        <div class="std_tab_pages_content">
          Tab 1 Content
        </div>
        <div class="std_tab_pages_content" style="display:none">
          Tab 2 Content
        </div>
        <div class="std_tab_pages_content" style="display:none">
          Tab 3 Content
        </div>
      </td>
    </tr>
  </tbody>
</table>
```

```
</tr>  
</tbody>  
</table>
```

## How the Default CSS Works

Without any CSS, the Tab Pages HTML will produce this result (table border added for clarity):



To start, you need to remove the list-style and margins from the UL tag:

```
ul.std_tab_pages_tabs
{
    list-style-type: none;
    margin: 0px;
}
```

Next you'll add some borders and background colors to the LI tags, use float:left to distribute them horizontally and apply a margin to separate them:

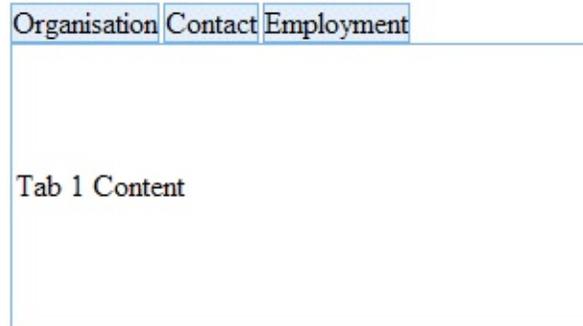
```
ul.std_tab_pages_tabs li
{
    border: 1px solid #7db0e5;
    background-color: #e2effa;
    color: black;
    white-space: nowrap;
    display: block;
    float: left;
    margin-right: 2px;
}
```

Finally, add a border around the content area:

```
.std_tab_pages_content_wrapper
{
    border: 1px solid #7db0e5;
```

```
padding: 2px;
}
```

Your table and list should now look like this:



Next, put a little spacing around the tab text and change the background color of the selected tab. You also need to remove the bottom border from the selected tab and extend the tab vertically to fill the space by increasing its bottom padding. (It is done this way because Internet Explorer draws the left and right borders unevenly if you try to change the color of the bottom border):

```
ul.std_tab_pages_tabs li a
{
    display: block;
    padding: 3px;
    text-decoration: none;
    font-weight: bold;
}
ul.std_tab_pages_tabs li.std_tab_active
{
    background-color: white;
    color: black;
    border-bottom: none;
    padding-bottom: 1px;
}
```

Finally you need to move the tabs down by the border width so that their bottom borders overlap the border of the content area. You can do this by making the table cell containing the tabs relatively positioned and shifting it down by the width of the border:

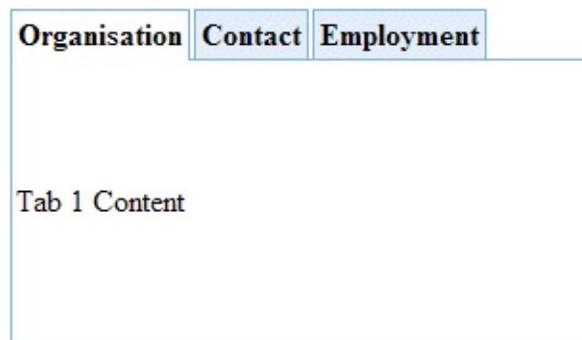
```
.std_tab_pages_top_tabs
{
```

```
position: relative;
top: 1px;
}
```

This cell shifting technique only works in Internet Explorer. In Firefox and Opera you need to move the UL block down by the border width (which does not work in IE):

```
ul.std_tab_pages_tabs
{
    position: relative;
    top: 1px;
}
```

The end result is this:



Some of the styles used only apply to tabs positioned along the top of the content area so you need to create some more styles with more specific selectors to ensure the properties are only applied to top aligned tabs:

```
.std_tab_pages_top_tabs ul
{
    position: relative;
    top: 1px;
}
.std_tab_pages_top_tabs ul li.std_tab_active
{
    border-bottom: none;
    padding-bottom: 1px;
}
```

## Adding your own CSS Styles

You can easily modify the appearance of your Tab Pages weblets by adding a few styles to a custom stylesheet to override the defaults. For example, the following will change the border and background colors:

```
.std_tab_pages_content_wrapper
{
    border: 1px solid #81a594;
}
ul.std_tab_pages_tabs li
{
    border: 1px solid #81a594;
    background-color: #cbcbb8;
}
```

Changing the border thickness will also require changing the selected tab padding and the amount that the tabs are shifted to overlap the content. These custom styles:

```
.std_tab_pages_content_wrapper
{
    border: 3px solid #81a594;
}
ul.std_tab_pages_tabs li
{
    border: 3px solid #81a594;
    background-color: #cbcbb8;
}
.std_tab_pages_top_tabs,
.std_tab_pages_top_tabs ul
{
    position: relative;
    top: 3px;
}
.std_tab_pages_top_tabs ul li.std_tab_active
{
    border-bottom: none;
    padding-bottom: 3px;
}
```

will produce this result:

Organisation	Contact	Employment
Street No and Name		17 Grantham Road
Suburb or Town		SEVEN HILLS
State and Country		NSW
Post / Zip Code		2147
Home Phone Number		718 1891
Business Phone Number		565 2341

Earlier Internet Explorer-only versions of the Tab Pages weblet would draw vertical text in left or right aligned tabs. Because this can only be done in Internet Explorer, the feature was removed from the \_v2 version. Left aligned tabs now look like this:

Organisation	Street No and Name	17 Grantham Road
Contact	Suburb or Town	SEVEN HILLS
Employment	State and Country	NSW
	Post / Zip Code	2147
	Home Phone Number	718 1891
	Business Phone Number	565 2341

If your target browser is Internet Explorer only then you can re-apply this effect with the *writing-mode* property:

```
.std_tab_pages_left_tabs ul li a
{
    writing-mode:tb-rl;
}
```

<b>Organisation</b>	Street No and Name	17 Grantham Road
	Suburb or Town	SEVEN HILLS
	State and Country	NSW
	Post / Zip Code	2147
	Home Phone Number	718 1891
	Business Phone Number	565 2341
<b>Contact</b>		
<b>Employment</b>		

If you prefer the text written from bottom to top, you can use the *filter* property to rotate it 180 degrees:

```
.std_tab_pages_left_tabs ul li a
{
  writing-mode:tb-rl;
  filter: flipv fliph;
}
```

<b>Organisation</b>	Street No and Name	17 Grantham Road
	Suburb or Town	SEVEN HILLS
	State and Country	NSW
	Post / Zip Code	2147
	Home Phone Number	718 1891
	Business Phone Number	565 2341
<b>Contact</b>		
<b>Employment</b>		

## Properties - Tab pages

The Tab pages weblet's properties are:

caption_field	image_field	tab_alignment
disable_if_true_field	listname	tab_image
formname	name	tab_image_alignment
content_height	pos_absolute_design	tab_image_height
content_width	selected_image_field	tab_image_width
hide_if	selected_tab_index	tab_selected_image
hide_if_true_field	selected_tab_index_field	tabs

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **tabs**

An XML nodeset that specifies the tab items to show. This is a system generated value set up when you drag the tab pages weblet onto the design view.

Do not directly edit the value shown. Instead, click the ellipsis button to open the tab items designer. Refer to Using the tab item designer for more information.

### **Default value**

`document(")/*//lxml:data/lxml:tab[@id='<unique id>']`

where the <unique id> is an automatically generated identifier.

### **Valid values**

Not Applicable (this value is system generated and should not be modified).

## **selected\_tab\_index**

Specifies the index of the tab that is to be initially selected when the web page is shown.

**Note:** Clicking on the tabs in the Design view alters this value to the index of the currently selected tab. Since you will do this routinely while designing your tab pages, you will need to remember to select the first tab again (or whichever tab you wish to be initially selected) before saving your work.

### **Default value**

1

### **Valid values**

Any integral value between 1 and the number of tabs (inclusive) or the name of a field that will contain the tab index at run-time.

If the value specified is less than 1 or greater than the number of tabs, then no tab is initially selected. In many applications of tab pages, this will not be desirable.

## **selected\_tab\_index\_field**

Specifies the name of a field that will contain the index of the selected tab. This is similar to the selected\_tab\_index property. Indeed you can specify a field name for the selected\_tab\_index property and its value will control the index of the tab that is initially selected. But using this property instead has the additional effect of updating the field with the index of the currently selected tab. If this field is posted back to the webroutine, then your program code can know which tab was last shown. You may, for example, wish to save this information and use it the next time the webroutine is loaded for the same user to show them the same tab they displayed on their previous visit.

### **Default value**

Blank.

### **Valid values**

The name of a field present in the web-map for your WebRoutine. (Usually you will specify a numeric field that is large enough to contain the highest tab number.) The field name should be specified in single quotes. Click the drop-down arrow next to this property to choose from a list of available field names.

## **tab\_alignment**

This property determines whether the tab buttons appear on the top, left, right or bottom of the tab pages.

### **Default value**

'top'

### **Valid values**

'top', 'left', 'right' or 'bottom'

## **tab\_image**

The path and file name, relative to the images virtual directory, of the image to show on each tab. This property applies to all tabs unless overridden in the tab item designer for individual tabs.

### **Default value**

'ball\_grn.gif' (this identifies a standard image shipped with LANSA)

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet. If you do not want to display an image on the tab you can enter an empty string (using two quote marks with nothing in-between).

## **tab\_selected\_image**

The path and file name, relative to the images virtual directory, of the image to show on each tab when the tab is selected. If specified, this property applies to all tabs unless overridden in the tab item designer for individual tabs.

### **Default value**

Blank – the image does not change when the tab item is selected.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **tab\_image\_height**

The height of tab button images.

### **Default value**

'14px'

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **tab\_image\_width**

The width of tab button images.

### **Default value**

'14px'

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **tab\_image\_alignment**

This property specifies whether the tab button images appear to the left or right of the caption.

### **Default value**

'left'

### **Valid values**

'left' or 'right'.

## **listname**

The name of a working list specified in the web\_map that will be used to dynamically override attributes of the tab items at run-time. Using the working list you can override any one or more of the following attributes:

- Tab caption (specify the working list field name in the caption\_field property)
- Tab image (specify the working list field name in the image\_field property)
- Tab selected image (specify the working list field name in the selected\_image\_field property)
- Tab hidden (specify the working list field name in the hide\_if\_true\_field property)
- Tab disabled (specify the working list field name in the disable\_if\_true\_field property).

To use this feature you must specify a working list in your web\_map (for \*OUTPUT or \*BOTH) that contains one or more fields corresponding to one or more of the attributes above that you wish to override. The RDML code in the webroutine should populate the list with the number of entries corresponding to the number of tab items with field values set as described for each of the properties mentioned above.

**Note:** You still need to use the tab items designer to create the required number of tab items and to set any attributes of the tab items that are not specified in the working list. Refer to Using the menu item designer for more information.

## **Default value**

Blank – no working list is used to override attributes of the tab items at run time. The tabs are displayed as defined in the tab items designer.

## **Valid values**

The name of a working list specified in the web-map for the WebRoutine. A list of available working lists can be selected from by clicking the corresponding dropdown button in the property sheet.

## **Example**

In this example a working list named A1OTABS will be used to override certain attributes of the tab items at run-time. The names of the fields

containing the overriding values for each tab item should be specified in one or more of the `caption_field`, `image_field`, `selected_image_field`, `hide_if_true_field` and/or `disable_if_true_field` properties.

 listname	'A10TABS'
--	-----------

## **caption\_field**

This property specifies the name of the field in the working list specified in the listname property that will override the tab caption for each tab item. This property is ignored if the listname property is not specified, but in any event is optional. Refer to the description of the listname property for further information.

### **Default value**

Blank – the tab caption is not overridden at run-time via the working list specified in the listname property.

### **Valid values**

The name of a field in the working list specified in the listname property that will contain the tab caption for each tab item. You can choose from a list of available fields by clicking the corresponding dropdown button in the property sheet.

## **image\_field**

This property specifies the name of the field in the working list specified in the listname property that will override the tab image filename (relative to the images virtual directory) for each tab item. This property is ignored if the listname property is not specified, but in any event is optional. Refer to the description of the listname property for further information.

### **Default value**

Blank – the tab image filename is not overridden at run-time via the working list specified in the listname property.

### **Valid values**

The name of a field in the working list specified in the listname property that will contain the tab image filename (relative to the images virtual directory) for each tab item. You can choose from a list of available fields by clicking the corresponding dropdown button in the property sheet.

## **selected\_image\_field**

This property specifies the name of the field in the working list specified in the listname property that will override the tab-selected image filename (relative to the images virtual directory) for each tab item. This property is optional and is ignored if the listname property is not specified. Refer to the description of the listname property for further information.

### **Default value**

Blank. The tab selected image filename is not overridden at run-time via the working list specified in the listname property.

### **Valid values**

The name of a field in the working list specified in the listname property that will contain the tab-selected image filename (relative to the images virtual directory) for each tab item. You can choose from a list of available fields from the corresponding dropdown list in the property sheet.

## **hide\_if\_true\_field**

This property specifies the name of the field in the working list specified in the listname property that will override the hidden state for each tab page. This property is ignored if the listname property is not specified, but in any event is optional. Refer to the description of the listname property for further information.

### **Default value**

Blank – the hidden state is not overridden at run-time via the working list specified in the listname property.

### **Valid values**

The name of a field in the working list specified in the listname property that will override the hidden state for each tab page. You can choose from a list of available fields by clicking the corresponding dropdown button in the property sheet.

In order to hide the tab page the webroutine must populate the field with one of these recognized true values: 'y', 'Y', 'true', or '1'.

## **disable\_if\_true\_field**

This property specifies the name of the field in the working list specified in the listname property that will override the disabled state for each tab page. This property is ignored if the listname property is not specified, but in any event is optional. Refer to the description of the listname property for further information.

### **Default value**

Blank – the disabled state is not overridden at run-time via the working list specified in the listname property.

### **Valid values**

The name of a field in the working list specified in the listname property that will override the disabled state for each tab page. You can choose from a list of available fields by clicking the corresponding dropdown button in the property sheet.

In order to disable the tab page the webroutine must populate the field with one of these recognized true values: 'y', 'Y', 'true', or '1'.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet. Note that hiding the weblet will hide the tab pages and all their contents.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:

hide_if	#STD_FLAG = 'Y'
---------	-----------------

When the property loses focus, the expression is shown as follows:

hide_if	key('field-value', 'STD_FLAG') = 'Y'
---------	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **content\_width**

The width of the content area of the weblet. That is, the width of the weblet excluding the tabs.

If you specify an empty string ("") the width will automatically adjust to the width of the content of the selected tab.

### **Default value**

'300px'

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **content\_height**

The height of the content area of the weblet. That is, the height of the weblet excluding the tabs.

If you specify an empty string ("") the height will automatically adjust to the height of the content of the selected tab.

### **Default value**

'150px'

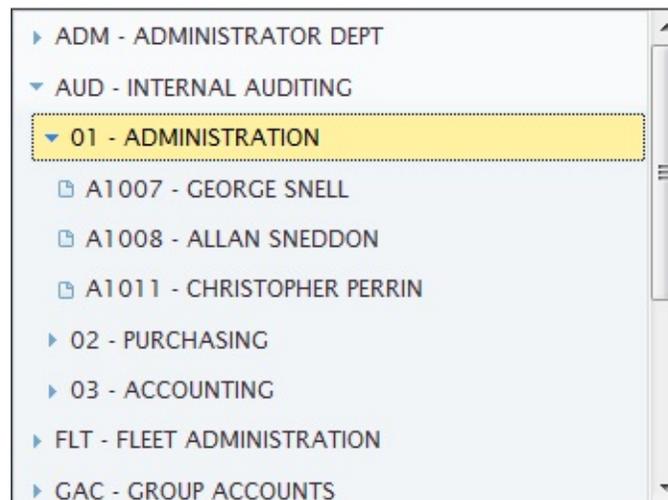
### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.1.31 Tree View (std\_treeview\_v2)

### QuickStart - Tree View Properties - Tree View

The Tree View weblet provides an expandable collapsible tree. It can be useful as a site navigation system or for visualizing complex hierarchical data.



It can also be used in conjunction with an <iframe> or the Navigation Panel weblet to display detail information in response to the selection of tree items (this replaces the TreeView Target weblet used in the previous version).

▶ ADMINISTRATOR DEPT	<b>Street No and Name</b>	22 Railway Parade,
▼ INTERNAL AUDITING	<b>Suburb or Town</b>	KOGARAH.
▼ ADMINISTRATION	<b>State and Country</b>	NSW.
▢ A1007 - GEORGE SNELL	<b>Post / Zip Code</b>	2160
▢ A1008 - ALLAN SNEDDON (highlighted)	<b>Home Phone Number</b>	476 2198
▢ A1011 - CHRISTOPHER PERRIN	<b>Business Phone Number</b>	630 8888
▶ PURCHASING	<b>Employee Salary</b>	450,000.04
▶ ACCOUNTING	<b>Start Date (DDMMYY)</b>	1/12/86
▶ FLEET ADMINISTRATION	<b>Termination Date (DDMMYY)</b>	0/00/00
▶ GROUP ACCOUNTS	<b>Monthly Salary</b>	37,500.00
▶ INFORMATION SERVICES		
▶ LEGAL DEPARTMENT		
▶ MANAGEMNT INFORMATIO		
▶ MARKETING DEPARTMENT		
▶ SALES & DISTRIBUTION		
▶ TRAVEL DEPARTMENT		

The Tree View is filled with data from a working list. Similar to the VL Tree View, the source data can be unlevelled, where each entry has an id and specifies the id of its parent, or levelled, where the tree builds itself from a sorted list based on key columns in the list.

When using unlevelled source data, the list can be configured to use Ajax to request child entries from the server when a branch is opened by the user.

## QuickStart - Tree View

[Unlevelled List](#)

[Levelled List](#)

[Using Ajax with an Unlevelled List](#) [Responding to item selection](#)

## Unlevelled List

In an unlevelled list each entry tells the Tree View exactly where it fits in the tree structure by specifying its parent ID. At a minimum, an unlevelled list must contain columns with the following data:

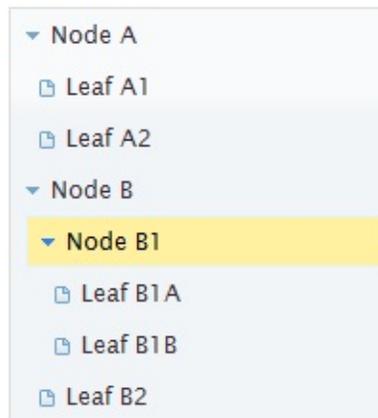
<b>Field</b>	<b>Tree View Property</b>	<b>Description</b>
ID	<a href="#">list_id_field</a>	A unique ID string to identify the entry.
Parent ID	<a href="#">list_parent_id_field</a>	The ID string of the parent entry. An empty string indicates a top level entry.
Caption	<a href="#">list_caption_field</a>	The text to display for the entry.

The Tree View processes the list entries in the supplied order and cannot add an entry to a parent that doesn't exist. It is your responsibility to ensure the list is sorted so that parent items come before their children and items at the same level are in display order. One way to do that is to add a depth column and a sequence column and sort the list by depth then sequence.

For example, a working list like this:

<b>ID</b>	<b>Parent ID</b>	<b>Caption</b>
1		Node A
2		Node B
3	1	Leaf A1
4	1	Leaf A2
5	2	Node B1
6	2	Leaf B2
7	5	Leaf B1A
8	5	Leaf B1B

Will produce a tree like this:



Note that, although the ID and ParentID columns contain numbers in this example, they must be text or character fields.

An unlevelled list can contain more columns that can be used to further customize the behavior or appearance of the corresponding entry.

Field	Tree View Property	Description
Image	<a href="#">list_image_field</a>	Specifies a path, relative to the images directory, to an image to be used for the entry's icon.
Open image	<a href="#">list_open_image_field</a>	Specifies a path, relative to the images directory, to an image to be used for the entry's icon when the entry is expanded.
Is Selected	<a href="#">list_is_selected_field</a>	Indicates that the initial state of the entry is selected. This should be a string field that will contain either nothing (not selected) or one of two values: 'true' or 'freeze'. Both values indicate that the entry is selected but differ in how

		<p>the Tree View behaves. If the value is 'true' the Tree View will perform whatever action it is configured to perform when a user clicks on the entry. In other words, it simulates the user selecting the entry. If the value is 'freeze' then the Tree View draws the item as selected but performs no other action.</p>
<p>Is Expanded</p>	<p><a href="#">list_is_expanded_field</a></p>	<p>Indicates that the initial state of the entry is expanded. This is ignored if the entry does not have children. This field can be a string or a numeric. Values of 'true', 'y' or 1 are treated as true, all other values are treated as false.</p>
<p>OnSelect WAM OnSelect Webroutine</p>	<p><a href="#">list_onselect_wamname_field</a> <a href="#">list_onselect_wrname_field</a></p>	<p>The name of a WAM/Webroutine that will be run when the item is clicked. If no value is supplied in these fields, the defaults specified by the onselect_wamname and onselect_wrname properties will be used.</p>

## Using Ajax with an Unlevelled List

When using an unlevelled list, it is possible to define just a partial list and to have the Tree View fetch further entries at a future time if and when they are needed. This technique can be useful for particularly large tree structures that might slow down the initial loading and rendering of the page.

To use this technique, you should add a "has children" field to your working list. This field (a single-byte field that should contain a Y or N) tells the Tree View to draw the item as an expandable node even if it has no children.

When the user expands a node that has no children the Tree View will call a webroutine to request a fresh copy of the working list containing new entries to add to the tree. The webroutine used for this purpose is defined with the `onexpand_wamname/onexpand_wrname` properties. The webroutine does not need a user interface as the Tree View will automatically request a JSON response from it.

Two fields and a list are sent to the webroutine.

<b>Field</b>	<b>Tree View Property</b>	<b>Description</b>
ID	<a href="#">onsubmit_id_field</a>	The ID of the expanded item
Level	<a href="#">onsubmit_level_field</a>	The level of the expanded item
Ancestors	<a href="#">onsubmit_ancestor_list</a>	A list containing the ancestors of the expanded item. The list will contain a single field which will be the same as the ID field of the working list used to create the tree.

While the properties above allow you to specify what fields the information is put into, you still need to define those fields in the `WEB_MAP` for the `onsubmit` webroutine.

## Levelled List

A levelled list is one where the ID and Caption for each level is represented by a different column. For example, the list may contain 3 columns: Department, Section, Name. The Tree View will automatically create "section" entries as children of "department" entries and "name" entries as children of "section" entries.

Two properties control how it does this:

**key\_fields** A comma delimited list of field names. These fields are used as keys for each level. As the list is processed, the key fields are compared, in order, with the key fields from the previous entry, if they change, a new tree entry is created at the key field level.

**display\_fields** A comma delimited list of field names. These fields contain the text to be displayed at the corresponding level.

For example, a working list like this:

<b>DEPTMENT</b>	<b>SECTION</b>	<b>EMPNO</b>	<b>DISPNAME</b>	<b>DEPTDESC</b>
ADM	01	A1001	BEN JONES	ADMINISTRA DEPT
AUD	01	A1007	GEORGE SNELL	INTERNAL AUDITING
AUD	01	A1008	ALLAN SNEDDON	INTERNAL AUDITING
AUD	01	A1011	CHRISTOPHER PERRIN	INTERNAL AUDITING
AUD	02	A1009	DAMIAN SNASHALL	INTERNAL AUDITING
AUD	03	A0907	ANNE MISS SIMPSON	INTERNAL AUDITING
AUD	03	A1010	WILLIAM PERRY	INTERNAL AUDITING

FLT	01	A1016	JACK TURNER	FLEET ADMINISTRA
FLT	02	A1003	Robert SMITHE	FLEET ADMINISTRA
FLT	03	A0090	FRED JOHN ALAN BLOGGS	FLEET ADMINISTRA
GAC	02	A1018	PAUL ZACHARIA	GROUP ACCOUNTS
INF	01	A1030	VALERIE TURNER	INFORMATIO SERVICES
INF	02	A1017	GARY NEAVE	INFORMATIO SERVICES
LEG	01	A1019	CHARLES DICKENS	LEGAL DEPARTMEN
LEG	03	A1023	DAVID REID	LEGAL DEPARTMEN
MIS	EI	A1031	JOHN BLAKE	MANAGEMN INFORMATIO
MKT	01	A1024	JOHN TAYLOR	MARKETING DEPARTMEN
MKT	02	A1022	KELLY THOMPSON	MARKETING DEPARTMEN
SD	ES	A1234	STEPHEN JACKSON	SALES & DISTRIBUTIO
TRVL	03	A1006	JACK SMITHERS	TRAVEL DEPARTMEN

With key\_fields and display\_fields set to:

--	--

key_fields	DEPARTMENT,SECTION,EMPNO
display_fields	DISPNAME,DEPTDESC,SECDESC

Would produce a tree like this:

- ▶ ADMINISTRATOR DEPT
- ▶ INTERNAL AUDITING
- ▼ FLEET ADMINISTRATION
- ▼ ADMINISTRATION
  - ▶ JACK TURNER
- ▶ PURCHASING
- ▶ ACCOUNTING
- ▶ GROUP ACCOUNTS
- ▶ INFORMATION SERVICES
- ▶ LEGAL DEPARTMENT
- ▶ MANAGEMNT INFORMATIO
- ▶ MARKETING DEPARTMENT
- ▶ SALES & DISTRIBUTION
- ▶ TRAVEL DEPARTMENT

## Responding to item selection

When a tree item is clicked, the Tree View will invoke a webroutine. If the [listtype](#) is 'unlevelled' then the webroutine specified in [list\\_onselect\\_wamname\\_field/list\\_onselect\\_wname\\_field](#) will be invoked. If the listtype is 'levelled' or an item specific webroutine has not been specified, the webroutine specified in [onselect\\_wamname/onselect\\_wname](#) will be invoked.

Two fields and a list are sent to the webroutine.

Field	Tree View Property	Description
ID	<a href="#">onsubmit_id_field</a>	The ID or key of the selected item
Level	<a href="#">onsubmit_level_field</a>	The level of the selected item
Ancestors	<a href="#">onsubmit_ancestor_list</a>	A list containing the ancestors of the selected item. The list will contain a single field which will be the same as the ID field of the working list used to create the tree.

While the properties above allow you to specify what fields the information is put into, you still need to define those fields in the WEB\_MAP for the onsubmit webroutine.

## Properties - Tree View

The Tree View weblet's properties are:

display_fields	list_is_expanded_field	onexpand_wname
folder_closed_image	list_is_selected_field	onselect_wamname
folder_open_image	list_onselect_wamname_field	onselect_wname
height	list_onselect_wname_field	onsubmit_ancestor_list
item_image	list_open_image_field	onsubmit_id_field
jQueryUI_node_icon	list_parent_id_field	onsubmit_level_field
key_fields	listname	pos_absolute
list_caption_field	listtype	target_window_name
list_haschildren_field	name	use_jQueryUI_theme
list_id_field	node_text_click	width
list_image_field	onexpand_wamname	

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. When using this weblet in conjunction with the Tree View Target (this would normally be the case), it is recommended that a name be entered, as the Tree View Target will be required to reference it. Using this name will be clearer than using the LANSAs-generated name.

**Default value**

`concat('oTree', ancestor-or-self::lxml:list/@name,position())` – this is the internal name given to the tree view by LANSAs.

**Valid values**

A name in single quotes.

**listname**

The name of the working list that contains the items used to populate the weblet. For best performance, your WEB\_MAP should define the list as \*JSON.

**Default value**

Blank. A valid list name must be entered.

**Valid values**

The name of a valid working list. A list of valid list names can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **listtype**

The type of data in the working list

## **Levelled**

In an unlevelled list, the data includes a unique ID for each item as well as the id of the item's parent. This type of list can be used to build a whole tree or it can be used to build an Ajax tree. In an Ajax tree the initial list contains a subset of the full tree (at least the visible items, possibly some sublevels). When the user attempts to open a branch that has no children the TreeView will execute a webrouting to obtain a new copy of the list containing new items to add to the tree.

Items in the TreeView will be created in list order. So, the items in the unlevelled list must be sorted such that parents occur before their children and children of a single parent occur in display order.

## **Unlevelled**

In a levelled list the TreeView infers the tree structure from the data. You provide the Tree View a list of column names (known as key fields) to use for folders and it automatically builds the tree structure by grouping items with the same key value under the same parent. The Tree View does this grouping by comparing the key fields of each entry with the key fields of the previous entry. It is your responsibility to ensure the list is correctly sorted into display order.

## **Default value**

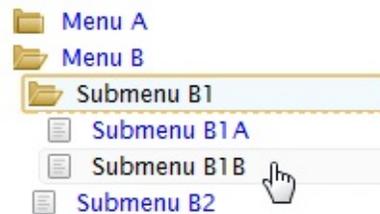
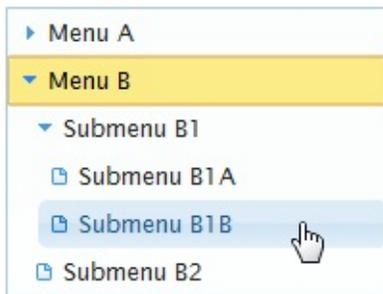
levelled

## **Valid values**

'levelled' or 'unlevelled'.

## use\_jQueryUI\_theme

The Tree view weblet is a jQuery-UI widget and will use the current jQuery-UI theme to produce its default appearance. If you are not using a jQuery-UI theme or don't want the treeview to use the theme, you can turn off the theme by setting this property to false().



A Tree View with the Redmond theme    A Tree View with no theme

### Default value

true()

### Valid values

true(), false() or a valid boolean expression.

## **jQueryUI\_node\_icon**

Specifies the jQuery-UI icon to be used for the node icon. Only valid if use\_jQueryUI\_theme is true().

### **Default value**

'triangle'

### **Valid values**

'folder', 'carat' or 'triangle'

## **folder\_closed\_image**

The path and file name, relative to the images directory, of an image that represents closed tree nodes. A blank value indicates that the Tree View should use its own default image.

### **Default value**

Blank

### **Valid values**

Blank or the path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **folder\_open\_image**

The path and file name, relative to the images directory, of an image that represents open tree nodes. A blank value indicates that the Tree View should use its own default image.

### **Default value**

Blank

### **Valid values**

Blank or the path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

**item\_image**

The path and file name, relative to the images directory, of an image to represent a leaf node of the tree. A blank value indicates that the Tree View should use its own default image.

**Default value**

Blank

**Valid values**

Blank or the path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **node\_text\_click**

Specifies what action to take when the user clicks on the text portion of an expandable node. Possible actions are to select the node, to toggle (expand/collapse) the node, or to do both.

### **Default value**

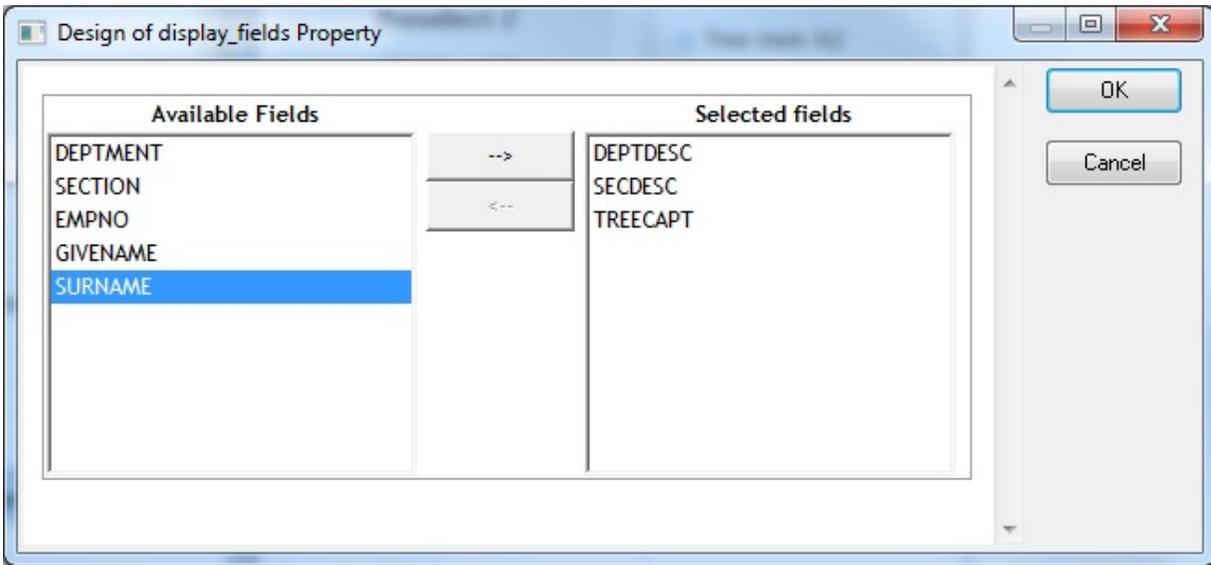
both

### **Valid values**

'select', 'toggle' or 'both'.

## key\_fields

If the listtype is 'levelled', this property specifies what fields the Tree View should use to determine item groups. The fields should be specified in depth order. You can open a custom property editor window by clicking the ellipse button in the property sheet.



## Default value

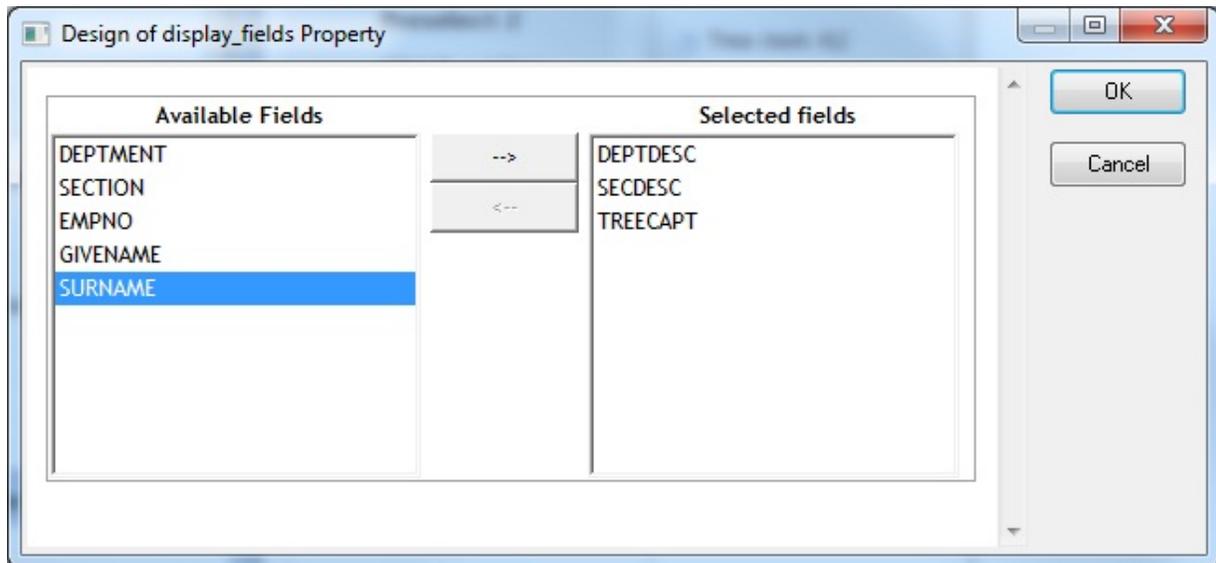
Blank. Must be specified for a levelled list.

## Valid values

A comma delimited list of field names.

## display\_fields

If the listtype is 'levelled' this property specifies what fields the Tree View should use to display the item text. The fields should be specified in depth order. You can open a custom property editor window by clicking the ellipse button in the property sheet.



### Default value

Blank. Must be specified for a levelled list.

### Valid values

A comma delimited list of field names.

## **list\_caption\_field**

The name of the field in the listname working list that contains tree item captions. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

Blank. A valid field name from the listname working list must be specified.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_image\_field**

The name of the field in the listname working list that contains a tree item's image path and file name, relative to the images directory. Leave blank if using default images. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

Default value

Blank. Default images are used.

## **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_open\_image\_field**

The name of the field in the listname working list that contains a tree item's image path and file name, relative to the images directory, that represents an expanded node. Leave blank if using default images. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

Blank. Default images are used.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_id\_field**

The name of the field in the listname working list that contains item IDs. This is the non-visible, unique identifier of the tree item that can be used to identify it when selected or expanded. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

\$list\_caption\_field. The field used to store the id information is the same field used for the caption.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

### **list\_onselect\_wamname\_field**

The name of the field in the listname working list that contains the name of the WAM whose Webroutine is to be invoked when a tree item is selected. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

Blank. The WAM specified by onselect\_wamname will used.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_onselect\_wrname\_field**

The name of the field in the listname working list that contains the name of the Webroutine that is to be invoked when a tree item is selected (the WAM containing the webroutine should be specified in list\_onselect\_wamname\_field). This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

Blank. The Webroutine specified by onselect\_wrname will invoked.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_haschildren\_field**

The name of the field in the listname working list that determines whether a tree item has child items. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

'STD\_CODE'

### **Valid values**

The name of a valid field, in single quotes, that will contain a:

'Y' (the tree item has child items) or an

'N' (the tree item does not have child items).

## **list\_is\_selected\_field**

The name of the field in the listname working list, the value of which will determine if a tree item should be selected when displayed. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

## **Default value**

Blank. Tree items cannot be pre-selected.

## **Valid values**

The name of the field in the working list that will contain a value of 'True' if an item in the tree should be selected. If set to 'Freeze', the item will be selected but the associated 'on select' action (if applicable) will not be triggered.

## **list\_is\_expanded\_field**

The name of the field in the working list that will control a tree item's expanded state. . This is only valid for unlevelled lists and will be ignored if listtype is "levelled"

### **Default value**

Blank. A tree item's expanded state cannot be controlled.

### **Valid values**

The name of the field in the working list that will contain a value of 'True' if an item should be expanded, and 'False' if it should not.

## **list\_parent\_id\_field**

The name of the field in the working list that will contain the identifier of the parent of the tree item. This is only valid for unlevelled lists and will be ignored if listtype is "levelled".

### **Default value**

Blank. If left blank, all items will be assumed to be top level items.

### **Valid values**

The name of the field in the working list that will contain the identifier of the parent of the tree item.

## **onselect\_wamname**

The name of the WAM whose Webroutine will be invoked when an item in the tree is selected. Individual items in an unlevelled list can override this using the `list_onselect_wamname_field` property.

### **Default value**

The current WAM.

### **Valid values**

The name of a WAM, in single quotes. A selection can be made from a list of known WAMs by clicking on the corresponding dropdown button in the property sheet.

## **onselect\_wrname**

The name of the Webroutine that will be invoked when an item in the tree is selected. Individual items in an unlevelled list can override this using the `list_onselect_wrname_field` property.

### **Default value**

The current Webroutine.

### **Valid values**

The name of a valid Webroutine, in single quotes. A selection can be made from a list of valid Webroutines by clicking on the corresponding dropdown button in the property sheet.

**onexpand\_wamname**

The name of the WAM whose Webroutine will be invoked when a node in the tree is expanded and has no children.

**Default value**

Blanks. The current WAM will be invoked.

**Valid values**

The name of a WAM, in single quotes. A selection can be made from a list of known WAMs by clicking on the corresponding dropdown button in the property sheet.

## **onexpand\_wrname**

The name of the Webroutine that will be invoked when a node in the tree is expanded and has no children.

### **Default value**

Blank. The current Webroutine is the default.

### **Valid values**

The name of a valid Webroutine, in single quotes. A selection can be made from a list of valid Webroutines by clicking on the corresponding dropdown button in the property sheet.

**onsubmit\_id\_field**

The name of the field that will be sent to the "On Expand" or "On Select" Webroutine containing the ID of the expanded or selected item. If not specified, the field specified by list\_id\_field will be used.

**Default value**

Blank. Use the value specified in list\_id\_field.

**Valid values**

The name of a valid input field. A selection can be made from a list of valid fields by clicking on the corresponding dropdown button in the property sheet.

### **onsubmit\_level\_field**

The name of the field that will be sent to the "On Expand" or "On Submit" Webroutine containing the level of the expanded or selected item. If not specified, the level will not be returned to the Webroutine.

### **Default value**

Blank. Do not send a level.

### **Valid values**

The name of a valid input field. A selection can be made from a list of valid fields by clicking on the corresponding dropdown button in the property sheet.

**onsubmit\_ancestor\_list**

The name of the working list returned to the Webroutine that will contain a list of parent identifiers for the currently selected or expanded tree item.

**Default value**

Blank. A list of parent identifiers is not passed to the Webroutine.

**Valid values**

The name of a valid working list, in single quotes. A selection can be made from a list of valid working lists by clicking on the corresponding dropdown button in the property sheet.

**target\_window\_name**

The name of the browser window, frame, iframe or Navigation Panel weblet that will be used to display the results of the 'on select' action.

**Default value**

Blank. The current page containing the Tree View will be replaced with the selected content.

**Valid values**

The name of a browser window, frame, iframe or Navigation Panel weblet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

Blank (the weblet expands to fill the space given to it).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the weblet on the web page.

**Default value**

Blank (the weblet expands to contain its content).

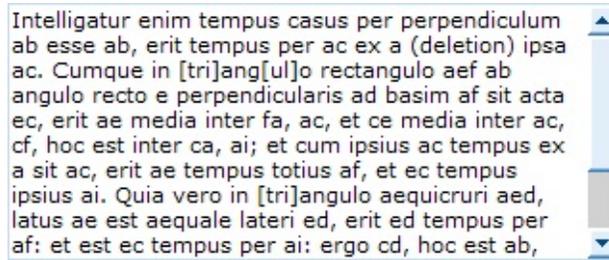
**Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.1.32 Memo using a field (std\_textarea)

[QuickStart - Memo using a field](#) [Properties - Memo using a field](#)

The Memo using a field (text area) weblet provides a text area for the display and input of long text values, possibly spanning multiple lines. It broadly corresponds to the <textarea> html element. The weblet looks like this:



## QuickStart - Memo using a field

It is common to use this weblet with long text fields. To use this weblet, open the XSL for your webroutine in the LANSAs Editor and follow the following steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Memo using a field weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.
4. Depending on the usage of the field in your application, you may need to set the value of the word\_wrap property.

**Note:** depending on the definition of the field in your webroutine, the generated XSL may already visualize the field using one of the std\_char or std\_varchar field visualization weblets. These weblets provide much of the functionality of the text area weblet with additional features.

## Properties - Memo using a field

The Memo using a field weblet's properties are:

class	maxlength	rows
cols	name	tab_index
disabled	onchange_script	value
height_design	pos_absolute	width_design
hide_if	read_only	word_wrap
keyboard_shift		

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this is the value of the field.

## Default value

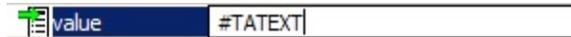
No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the text area and/or that is used to receive the text from the text area.

## Valid values

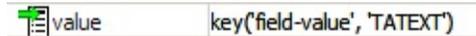
Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how a field name is entered for the value property when the weblet visualizes a field:



When the property loses focus, the field name is shown as follows:



**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

## **keyboard\_shift**

The keyboard shift for the input field.

### **Default value**

The keyboard shift of the field with this weblet visualization. Blank otherwise.

### **Valid values**

Char and String data types: ' ', 'W', 'J', 'E', 'O' and 'U'

Alpha data type: ' ', 'X', 'A', 'N', "W", 'T', 'D', 'M', 'J', 'E' and 'O'

The keyboard shift is currently only used to validate DBCS fields.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

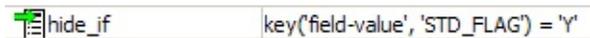
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is '#STD\_FLAG = 'Y''. To the left of the property name is a small icon representing a document or list.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is shown in a light grey box. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. To the left of the property name is a small icon representing a document or list.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSAs Editor. This resulted in the value shown for the pos\_absolute\_design property.

 pos_absolute_design	'position:absolute;left: 324pt; top: 162.72pt;'
---	---

**width\_design**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property value if required.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height\_design**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

Blank (weblet uses its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

**rows**

The number of visible rows in the text area. This property sets the height of the weblet such that the specified number of rows or lines of text will be visible.

If the height\_design property is specified, it takes precedence and the rows property is ignored.

**Default value**

10

**Valid values**

A number that specifies the number of rows.

## **cols**

The number of columns in the text area. This property sets the width of the weblet, based on the average character width for the font used, such that approximately the specified number of columns of text will be visible.

If the `width_design` property is specified, it takes precedence and the `cols` property is ignored.

### **Default value**

50

### **Valid values**

A number that specifies the number of columns.

## **word\_wrap**

This property specifies how the weblet should handle word-wrapping when typing text.

### **Default value**

Blank.

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

- 'soft'** Text is displayed with word wrapping and submitted without carriage returns and line feeds.
- 'hard'** Text is displayed with word wrapping and submitted with soft returns and line feeds.
- 'off'** Word wrapping is disabled. The lines appear exactly as the user types them.

## **class**

The Cascading Style Sheet (CSS) class name for the weblet.

### **Default value**

The name of the shipped class for the weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

### **Example**

The example shown sets the CSS class name to 'bold':



## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

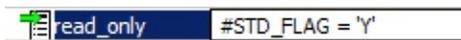
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''. The dropdown menu has a small icon to its left.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''. The dropdown menu has a small icon to its left.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **onchange\_script**

JavaScript code to be run when the text area loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## 8.2 Charting Webllets

The charting webllets provide you a means to visualize data. They are useful to compare distribution of values, trends and proportional part-to-whole information.

The charting webllets use the Google Image Chart API. The webllets provide you an easy way to request chart images from Google.

**Note: The image charts is a service provided by Google so you need to abide by Google's Chart Usage Policy:**

**Google Chart Usage Policy**

There's no limit to the number of calls per day you can make to the Google Chart API. However, Google reserves the right to block any use that it regards as abusive. If you think your service may risk being blocked, contact Google.

There are three types of charts you can create using the Google Charting webllets:

<b>Webllet name</b>	<b>Description</b>
<a href="#">Google Bar Chart (std_gbar_chart)</a>	Single and multiple data series. You can create vertical, horizontal, grouped, stacked, and overlapped bar charts.
<a href="#">Google Line Chart (std_gline_chart)</a>	Single and multiple data series. You can create various types of line charts and put markers on data points.
<a href="#">Google Pie Chart</a>	Simple and 3D pie charts for single data series. Use concentric pie charts for multiple data series.

## **8.2.1 Common Chart Topics**

- [Chart Data](#)
- [Chart Colors](#)
- [Chart Title, Label and Legends](#)
- [Chart Margins](#)

## Chart Data

You provide the data in a JSON list. See [JSON Lists](#). The list columns must all be numeric. Each column is a data value in the series. Each list row is a data series.

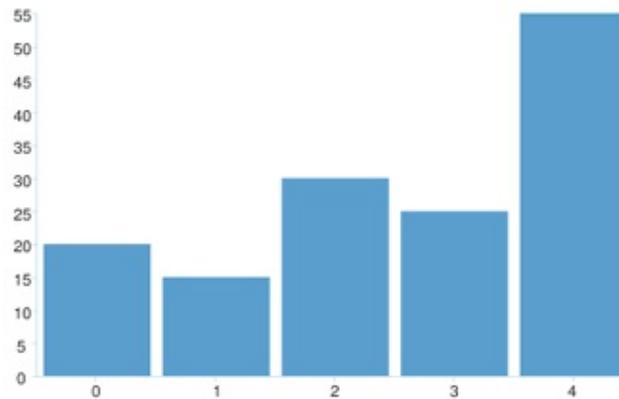
The following examples use a Bar chart but the concept applies equally to the other charts.

- [Samples](#)
- [Transposing List Rows and Columns](#)

## Samples

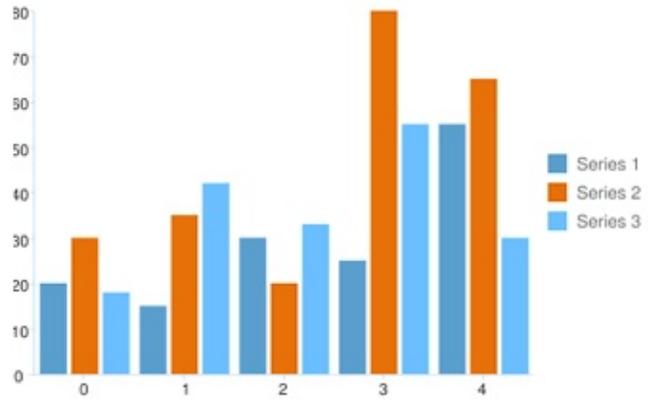
Sample **single** data series list

Column 1	Column 2	Column 3	Column 4	Column 5
20	15	30	25	55



Sample **multiple** data series list (Three data series)

Column 1	Column 2	Column 3	Column 4	Column 5
20	15	30	25	55
30	35	20	80	65
18	42	35	55	30



## Transposing List Rows and Columns

It is easier to build lists where the data values are represented by the list rows rather than the column. Use the transpose property in the charts to transpose the list rows and columns.

For example if you have the following list:

<b>Column 1</b>
20
15
30
25
55

If you set the transpose property to true, it is as if you were using the following list:

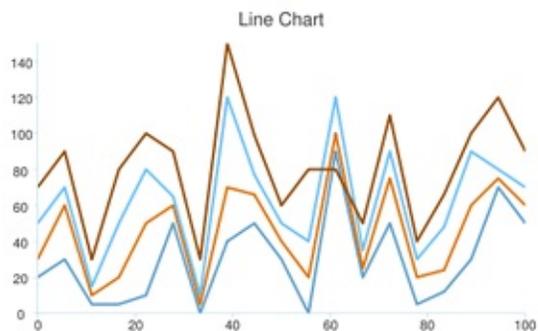
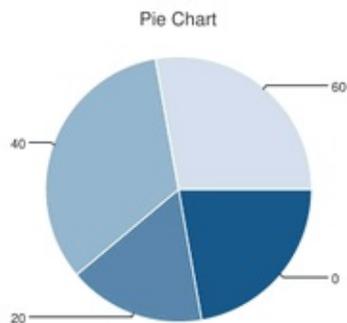
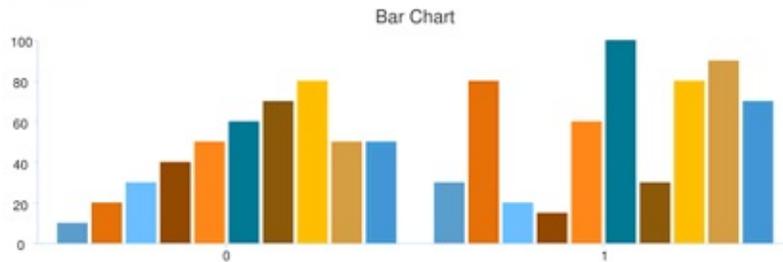
<b>Column 1</b>	<b>Column 2</b>	<b>Column 3</b>	<b>Column 4</b>	<b>Column 5</b>
20	15	30	25	55

## Chart Colors

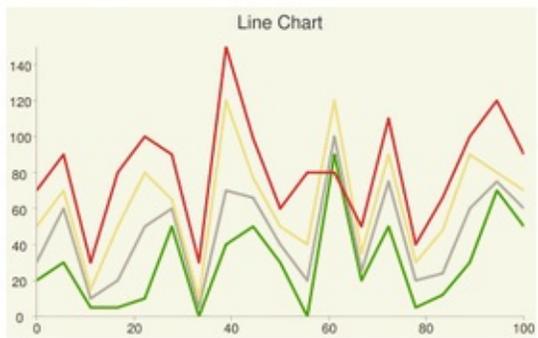
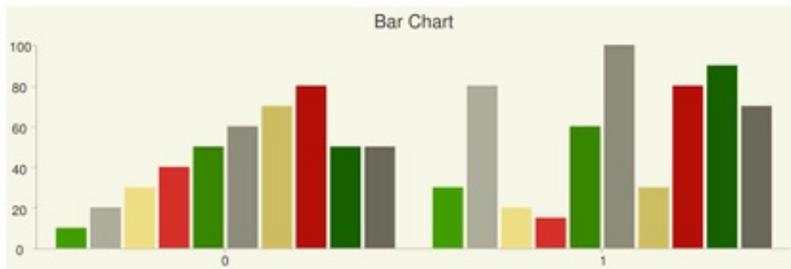
If you don't designate colors, the charts use color schemes compatible with the theme in use. See [Theming WAMs](#). This includes series colors, title, labels, range labels, legends and background colors.

## Examples

### Redmond

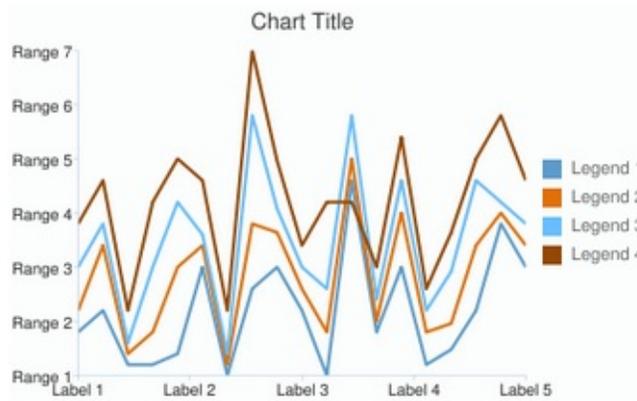


## South Street



## Chart Title, Label and Legends

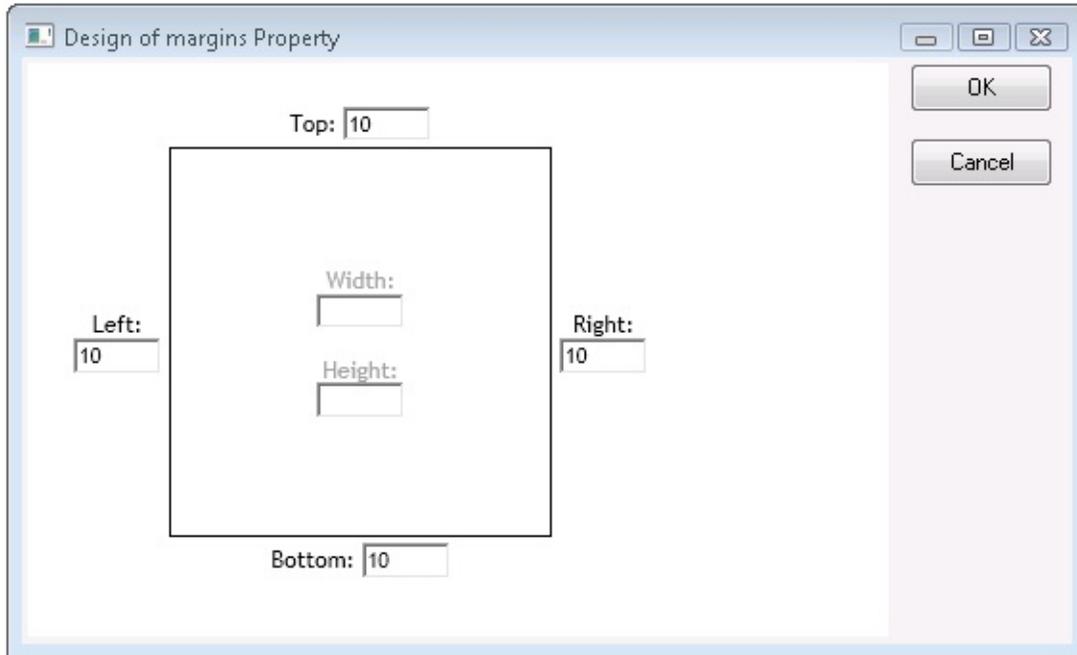
You can add a title, custom labels and legends to all charts. The property customizers allow you to enter text values or use multilingual text variables.



## Chart Margins

You can specify the size of the chart's margins, in pixels. Margins are calculated inward from the specified chart size (width x height); Note that the margins don't increase the total chart size, but rather shrinks the chart area, if necessary.

Click on the margins property customizer to enter margins. If you don't enter margins, Google will determine them automatically based on your chart size.



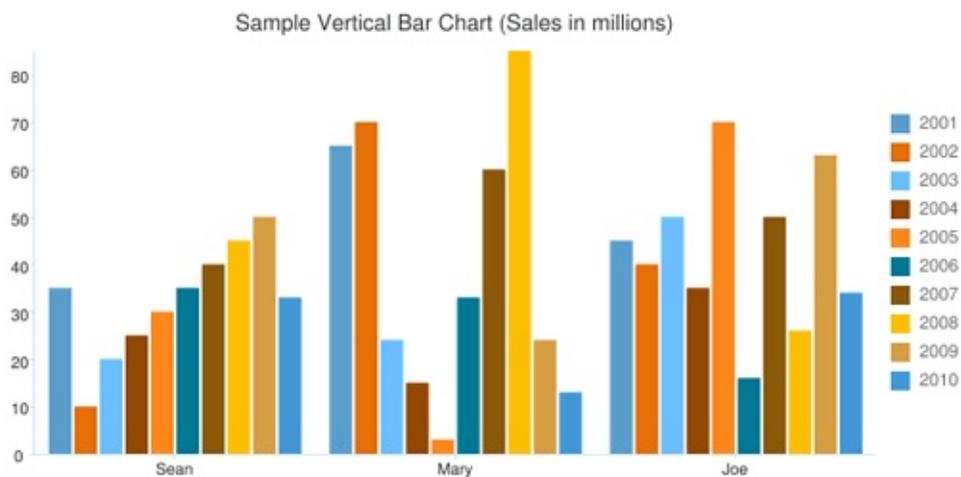
The image shows a dialog box titled "Design of margins Property". It contains a central square representing a chart area. Surrounding this square are five input fields for margin values: "Top: 10", "Left: 10", "Right: 10", and "Bottom: 10". In the center of the square, there are two empty input fields labeled "Width:" and "Height:". To the right of the dialog box are "OK" and "Cancel" buttons.

## 8.2.2 Google Bar Chart (std\_gbar\_chart)

### [QuickStart – Google Bar Chart Properties – Google Bar Chart](#)

Bar charts visualize data points as vertical or horizontal bars that are proportional to the data value. You can create, vertical, horizontal, grouped, stacked, and overlapped bar charts.

Bar charts are good for side-by-side comparison and visualizing trends in a small number of discrete data points.



## **QuickStart – Google Bar Chart**

To create a bar chart, you need to create a webroutine that specifies the a JSON list with the data values in its WEB\_MAP as \*OUTPUT.

1. Click on the Weblets tab, select Charting Weblets from the drop-down list near the top and locate the Google Bar Chart weblet.
2. Drag the Google Bar Chart weblet over the designer and release the left-mouse button. A placeholder image will appear on the designer.
3. Set the listName property with the list name in your WEB\_MAP.
4. Use the transpose property if you want to transpose rows and column
5. Set your chart title, labels and legends as required.

## Properties – Google Bar Chart

axesColor	legendMargins	rangeLabelsFontSize
barWidth	legendOrder	seriesColor
bgColor	legendPos	spaceBetweenBars
chartType	legendText	spaceBetweenGroups
height	listName	titleColor
hide_if	margins	titleFontSize
labels	name	titleText
labelsColor	pos_absolute	transpose
labelsFontSize	rangeLabels	width
legendColor	rangeLabelsColor	
legendFontSize		

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

A default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **chartType**

The type of bar chart. Bars can be horizontal or vertical and the bars can be stacked (atop or in front of each other) or grouped.

### **Default value**

Vertical bar chart with grouped bars

### **Valid values**

Horizontal bar chart with stacked bars

Vertical bar chart with stacked bars(atop)

Vertical bar chart with stacked bars (in front)

Horizontal bar charts with grouped bars

Vertical bar chart with grouped bars

<p>The horizontal and vertical bar charts with stacked bars only work for lists with positive values. If your data has negative values, use either the horizontal or vertical bar charts with grouped bars.</p>
---

**listName**

Name of the JSON list containing the data to be visualized in the bar chart. Each column in the list will be a bar. Each row in the list is a series.

**Default value**

No default value. A list name is required.

**Valid values**

The name of a JSON list with valid data values (numeric series).

## **transpose**

If true, the list columns and rows are transposed so that each column becomes a series.

### **Default value**

False

### **Valid values**

false() or true()

**labels**

Labels to use in the base axis.

**Default value**

The chart will assign sequence numbers to each data value or data series group.

**Valid values**

A list of strings or multilingual text variables.

## **labelsColor**

The color of the labels text.

### **Default value**

Theme: Uses a color compatible with the theme (if a theme is present).

### **Valid values**

Color must be in RRGGBB hexadecimal format.

## **labelsFontSize**

Font size of the chart labels, in pixels.

### **Default value**

If not provided, Google sets it automatically

### **Valid values**

A numeric value in pixels.

**rangeLabels**

Labels to use in the range axis.

**Default value**

The range of data values is used to assign range values at intervals in the range axis.

**Valid values**

A list of strings or multilingual text variables.

## **rangeLabelsColor**

The color of the range labels text.

### **Default value**

Theme: Uses a color compatible with the theme (if a theme is present).

### **Valid values**

Color must be in RRGGBB hexadecimal format.

## **rangeLabelsFontSize**

Font size of the chart range labels, in pixels.

### **Default value**

If not provided, Google sets it automatically

### **Valid values**

A numeric value in pixels.

## **barWidth**

[Optional] Bar width, in pixels. Enter 'spaceBetweenBars' and 'spaceBetweenGroups' also in absolute values, in pixels. Bars can be clipped if the chart isn't wide enough. Use 'auto-absolute' or 'auto-relative' to have bars resized so that all bars will fit in the chart.

Absolute values: Values are given in absolute units (or default absolute values, if not specified). Bars will be resized so that all bars will fit in the chart

Relative values: Values are given in relative units (or default relative values, if not specified) Relative units are floating point values compared to the bar width, where the bar width is 1.0: for example, 0.5 is half the bar width, 2.0 is twice the bar width. Bars can be clipped if the chart isn't wide enough.

### **Default value**

23 pixels (absolute value)

### **Valid values**

Numeric value (in pixels) or 'auto-absolute' or 'auto-relative'

## **spaceBetweenBars**

[Optional] Space between bars. Must be entered in absolute values, in pixels, if the bar width has been specified in pixels or 'auto-absolute'. If bar width is 'auto-relative', this is a floating point value where 1.0 is the bar width.

### **Default value**

4 pixels for absolute values, or 4/23 for relative values.

### **Valid values**

A numeric value.

## **spaceBetweenGroups**

[Optional] Space between Groups. Must be entered in absolute values, in pixels, if the bar width has been specified in pixels or 'auto-absolute'. If bar width is 'auto-relative', this is a floating point value where 1.0 is the bar width.

### **Default value**

8 pixels for absolute values, or 8/23 for relative values.

### **Valid values**

A numeric value

**seriesColor**

Colors to be used for each series. Colors must be in RRGGBB hexadecimal format.

**Default value**

Theme: Uses colors (one for each data series, up to a maximum of 10 data series) compatible with the theme (if a theme is present).

**Valid values**

A comma separated list of colors. Colors must be in RRGGBB hexadecimal format.

**bgColor**

Chart background color. Color must be in RRGGBB hexadecimal format.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

Chart width, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

400 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**height**

Chart height, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

250 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**titleText**

The chart title. Use a pipe character (|) to indicate line breaks.

**Default value**

No title

**Valid values**

Text string or multilingual text variable.

**titleColor**

The color of the chart title.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**titleFontSize**

Font size of the chart title, in points.

**Default value**

If not provided, Google sets it automatically.

**Valid values**

Numeric value, in points.

**axesColor**

The color of the chart axes.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

## **margins**

Minimum margin size around the chart area, in pixels (left, right, top, bottom). Increase this value to include some padding to prevent axis labels from bumping against the borders of the chart.

### **Default value**

The margins are by default whatever is left over after the chart size is calculated.

### **Valid values**

Comma separated numeric values in pixels (left, right, top and bottom).

**legendText**

Optional. The text for the legend entries. Each label applies to the corresponding series in the data array. If you do not specify this parameter, the chart will not get a legend. There is no way to specify a line break in a label. The legend will typically expand to hold your legend text, and the chart area will shrink to accommodate the legend.

**Default value**

No legends

**Valid values**

Comma separated list of strings or multilingual text variables.

## **legendPos**

Optional. The position of the legend. You can add an 's' to any value if you want empty legend entries in legendText to be skipped in the legend.

### **Default value**

Legend to the right of the chart, legend entries in a vertical column.

### **Valid values**

- Right of chart, in vertical column
- Bottom of chart, in horizontal row
- Bottom of chart, in vertical column
- Top of chart, in horizontal row
- Top of chart, in vertical column
- Left of chart, in vertical column

## **legendOrder**

Optional. The order of the legend. Display in order (Default for vertical legends) shows labels in the order they were entered. Reverse order is useful in stacked bar charts to show the legend in the same order as the bars appear.

Automatic ordering (Default for horizontal legends): roughly means sorting by length, shortest first, as measured in 10 pixel blocks. When two elements are the same length (divided into 10 pixel blocks), the one listed first will appear first.

### **Default value**

Display in order

### **Valid values**

Display in order

Reverse order

Automatic ordering

**legendColor**

The color of the legend text.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**legendFontSize**

Font size of the legend text, in points.

**Default value**

Automatically set by Google.

**Valid values**

A numeric value in points.

## **legendMargins**

Optional. Width of the margin around the legend (width, height), in pixels. Use this to avoid having the legend bump up against the chart area or the edges of the image.

### **Default value**

Automatically set by Google.

### **Valid values**

Comma separated numbers (width, height) in pixels.

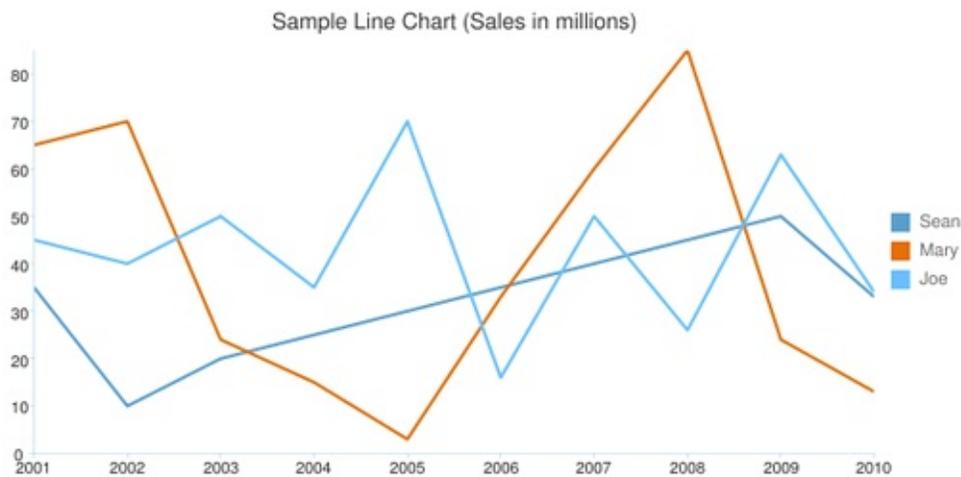
## 8.2.3 Google Line Chart (std\_gline\_chart)

### [QuickStart – Google Line Chart Properties – Google Line Chart](#)

Line charts visualize data points joined together by a line.

You can line charts in which your data values represent the points along the range axis distributed at even intervals in the horizontal axis or you provide pairs of data series representing the x,y pairs.

Line charts are good for visualizing trends in a large number of discrete data points.



## QuickStart – Google Line Chart

To create a line chart, you need to create a webroutine that specifies the a JSON list with the data values in its WEB\_MAP as \*OUTPUT.

1. Click on the Weblets tab, select Charting Weblets from the drop-down list near the top and locate the Google Line Chart weblet.
2. Drag the Google Line Chart weblet over the designer and release the left-mouse button. A placeholder image will appear on the designer.
3. Set the listName property with the list name in your WEB\_MAP.
4. Use the transpose property if you want to transpose rows and column
5. Set your chart title, labels and legends as required.

## Properties – Google Line Chart

axesColor	legendMargins	pos_absolute
bgColor	legendOrder	rangeLabels
chartType	legendPos	rangeLabelsColor
height	legendText	rangeLabelsFontSize
hide_if	lineThickness	seriesColor
labels	listName	titleColor
labelsColor	margins	titleFontSize
labelsFontSize	markerColor	titleText
legendColor	markerType	transpose
legendFontSize	name	width

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

A default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **chartType**

The type of line chart. Line charts can be simple line charts, sparklines (no axes) and line chart with x,y coordinates for each point.

### **Default value**

Line Chart

### **Valid values**

Line Chart

Line Chart with no Axis (Sparklines)

Line Chart with x,y Coordinates for each Point

**listName**

Name of the JSON list containing the data to be visualized in the line chart. Each column in the list will be a data point. Each row in the list is a series.

**Default value**

No default value. A list name is required.

**Valid values**

The name of a JSON list with valid data values (numeric series).

## **transpose**

If true, the list columns and rows are transposed so that each column becomes a series.

### **Default value**

False

### **Valid values**

false() or true()

**labels**

Labels to use in the base axis.

**Default value**

The chart will assign sequence numbers to each data value or data series group.

**Valid values**

A list of strings or multilingual text variables.

## **labelsColor**

The color of the labels text.

### **Default value**

Theme: Uses a color compatible with the theme (if a theme is present).

### **Valid values**

Color must be in RRGGBB hexadecimal format.

## **labelsFontSize**

Font size of the chart labels, in pixels.

### **Default value**

If not provided, Google sets it automatically

### **Valid values**

A numeric value in pixels.

**rangeLabels**

Labels to use in the range axis.

**Default value**

The range of data values is used to assign range values at intervals in the range axis.

**Valid values**

A list of strings or multilingual text variables.

## **rangeLabelsColor**

The color of the range labels text.

### **Default value**

Theme: Uses a color compatible with the theme (if a theme is present).

### **Valid values**

Color must be in RRGGBB hexadecimal format.

## **rangeLabelsFontSize**

Font size of the chart range labels, in pixels.

### **Default value**

If not provided, Google sets it automatically

### **Valid values**

A numeric value in pixels.

**seriesColor**

Colors to be used for each series. Colors must be in RRGGBB hexadecimal format.

**Default value**

Theme: Uses colors (one for each data series, up to a maximum of 10 data series) compatible with the theme (if a theme is present).

**Valid values**

A comma separated list of colors. Colors must be in RRGGBB hexadecimal format.

**bgColor**

Chart background color. Color must be in RRGGBB hexadecimal format.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

## lineThickness

The thickness of the line in pixels.

## Default value

thin

## Valid values

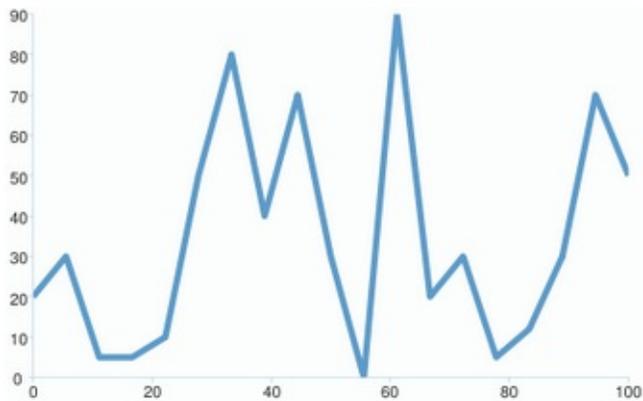
thin

medium

thick

## Example

Line chart with medium line thickness:



## markerType

Select a marker type to add markers for data points in the line chart.

## Default value

None

## Valid values

None

Cross

Diamond

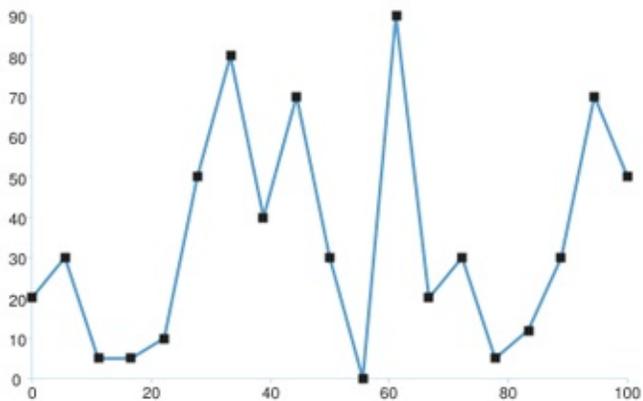
Circle

Square

An X

## Example

Line chart with square markers:



**markerColor**

The color for the markers.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

Chart width, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

400 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**height**

Chart height, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

250 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**titleText**

The chart title. Use a pipe character (|) to indicate line breaks.

**Default value**

No title

**Valid values**

Text string or multilingual text variable.

**titleColor**

The color of the chart title.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**titleFontSize**

Font size of the chart title, in points.

**Default value**

If not provided, Google sets it automatically.

**Valid values**

Numeric value, in points.

**axesColor**

The color of the chart axes.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

## **margins**

Minimum margin size around the chart area, in pixels (left, right, top, bottom). Increase this value to include some padding to prevent axis labels from bumping against the borders of the chart.

### **Default value**

The margins are by default whatever is left over after the chart size is calculated.

### **Valid values**

Comma separated numeric values in pixels (left, right, top and bottom).

**legendText**

Optional. The text for the legend entries. Each label applies to the corresponding series in the data array. If you do not specify this parameter, the chart will not get a legend. There is no way to specify a line break in a label. The legend will typically expand to hold your legend text, and the chart area will shrink to accommodate the legend.

**Default value**

No legends

**Valid values**

Comma separated list of strings or multilingual text variables.

## **legendPos**

Optional. The position of the legend. You can add an 's' to any value if you want empty legend entries in legendText to be skipped in the legend.

### **Default value**

Legend to the right of the chart, legend entries in a vertical column.

### **Valid values**

- Right of chart, in vertical column
- Bottom of chart, in horizontal row
- Bottom of chart, in vertical column
- Top of chart, in horizontal row
- Top of chart, in vertical column
- Left of chart, in vertical column

## **legendOrder**

Optional. The order of the legend. Display in order (Default for vertical legends) shows labels in the order they were entered. Reverse order is useful in stacked bar charts to show the legend in the same order as the bars appear.

Automatic ordering (Default for horizontal legends): roughly means sorting by length, shortest first, as measured in 10 pixel blocks. When two elements are the same length (divided into 10 pixel blocks), the one listed first will appear first.

### **Default value**

Display in order

### **Valid values**

Display in order

Reverse order

Automatic ordering

**legendColor**

The color of the legend text.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**legendFontSize**

Font size of the legend text, in points.

**Default value**

Automatically set by Google.

**Valid values**

A numeric value in points.

## **legendMargins**

Optional. Width of the margin around the legend (width, height), in pixels. Use this to avoid having the legend bump up against the chart area or the edges of the image.

### **Default value**

Automatically set by Google.

### **Valid values**

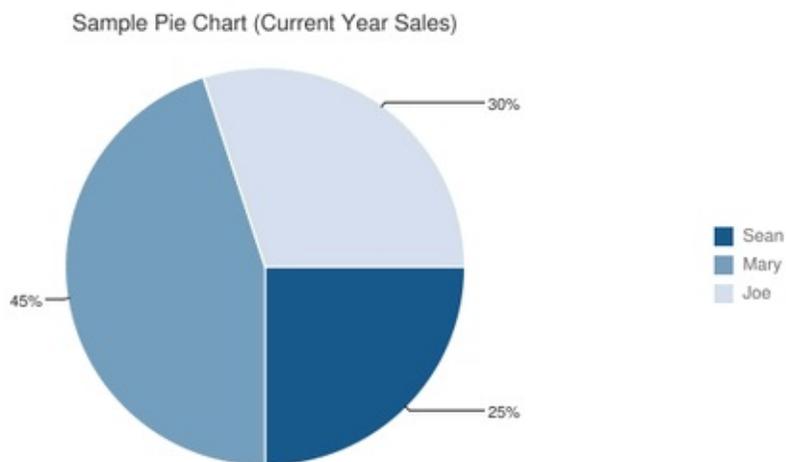
Comma separated numbers (width, height) in pixels.

## 8.2.4 Google Pie Chart (std\_gpie\_chart)

### QuickStart – Google Pie Chart Properties – Google Pie Chart

Pie charts are good to visualize part-to-whole comparisons, like market share for products, sales per product line, etc.

You can create simple or 3D pie charts for single data series. You can use concentric pie charts to visualize multiple data series with each data series showing as a concentric ring.



## **QuickStart – Google Pie Chart**

1. Click on the Weblets tab, select Charting Weblets from the drop-down list near the top and locate the Google Pie Chart weblet.
2. Drag the Google Pie Chart weblet over the designer and release the left-mouse button. A placeholder image will appear on the designer.
3. Set the listName property with the list name in your WEB\_MAP.
4. Use the transpose property if you want to transpose rows and column
5. Set your chart title, labels and legends as required.

## Properties – Google Pie Chart

bgColor	legendFontSize	pos_absolute
chartType	legendMargins	rotation
height	legendOrder	seriesColor
hide_if	legendPos	titleColor
labels	legendText	titleFontSize
labelsColor	listName	titleText
labelsFontSize	margins	transpose
legendColor	name	width

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

A default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **chartType**

The type of pie chart. There are three general types of pie charts that you can create: flat, concentric, or 3D.

## **Default value**

Two dimensional pie chart

## **Valid values**

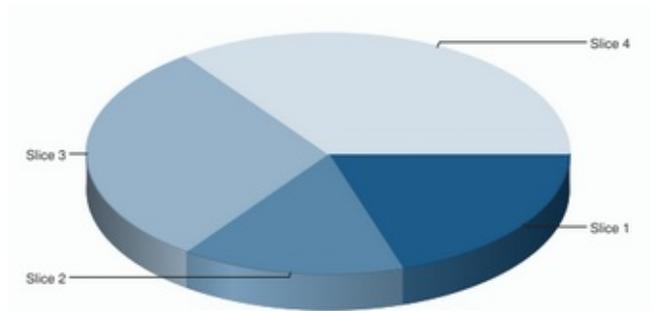
Two dimensional pie chart

Three-dimensional pie chart

Concentric pie chart

## **Example**

Three-dimensional pie chart:



**listName**

Name of the JSON list containing the data to be visualized in the pie chart. Each column in the list will be a slice in the pie chart. For concentric pie charts, each row in the list shows as a concentric ring in the pie chart.

**Default value**

No default value. A list name is required.

**Valid values**

The name of a JSON list with valid data values (numeric series).

## **transpose**

If true, the list columns and rows are transposed so that each column becomes a series.

### **Default value**

False

### **Valid values**

false() or true()

**labels**

Labels to use for the pie chart slices.

**Default value**

The chart will assign sequence numbers to each slice.

**Valid values**

A list of strings or multilingual text variables.

## **labelsColor**

The color of the labels text.

### **Default value**

Theme: Uses a color compatible with the theme (if a theme is present).

### **Valid values**

Color must be in RRGGBB hexadecimal format.

## **labelsFontSize**

Font size of the chart labels, in pixels.

### **Default value**

If not provided, Google sets it automatically

### **Valid values**

A numeric value in pixels.

## rotation

By default, the first series is drawn starting at 3:00, continuing clockwise around the chart. Select a different value to start at a different position.

### Default value

3:00

### Valid values

0:00

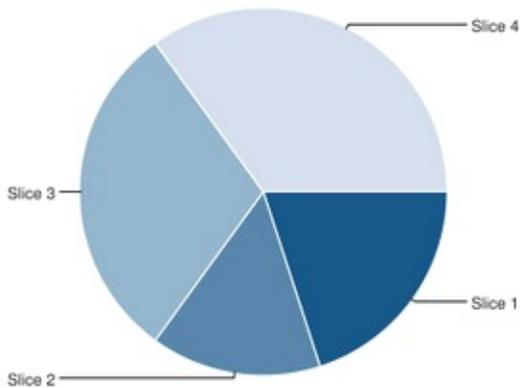
3:00

6:00

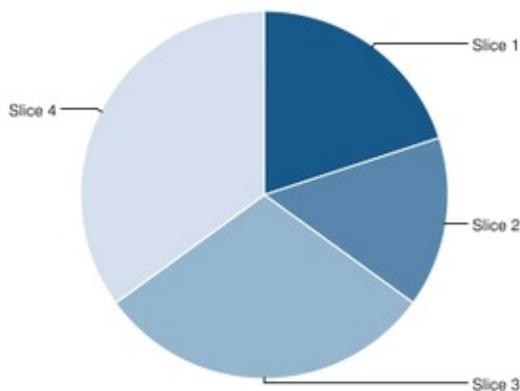
9:00

### Example

Default rotation (3:00 o'clock):



Rotation at 0:00 o'clock:



**seriesColor**

Colors to be used for each slice. Colors must be in RRGGBB hexadecimal format.

**Default value**

Theme: Uses a monochromatic set of colors, starting with a color compatible with the main theme color (or darker, if necessary).

**Valid values**

A comma separated list of colors. Colors must be in RRGGBB hexadecimal format.

**bgColor**

Chart background color. Color must be in RRGGBB hexadecimal format.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

Chart width, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

400 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**height**

Chart height, in pixels. Maximum value is 1,000. Width x height cannot exceed 300,000.

**Default value**

250 pixels

**Valid values**

Numeric value up to 1,000. Width x height cannot exceed 300,000.

**titleText**

The chart title. Use a pipe character (|) to indicate line breaks.

**Default value**

No title

**Valid values**

Text string or multilingual text variable.

**titleColor**

The color of the chart title.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**titleFontSize**

Font size of the chart title, in points.

**Default value**

If not provided, Google sets it automatically.

**Valid values**

Numeric value, in points.

## **margins**

Minimum margin size around the chart area, in pixels (left, right, top, bottom). Increase this value to include some padding to prevent axis labels from bumping against the borders of the chart.

### **Default value**

The margins are by default whatever is left over after the chart size is calculated.

### **Valid values**

Comma separated numeric values in pixels (left, right, top and bottom).

**legendText**

Optional. The text for the legend entries. Each label applies to the corresponding series in the data array. If you do not specify this parameter, the chart will not get a legend. There is no way to specify a line break in a label. The legend will typically expand to hold your legend text, and the chart area will shrink to accommodate the legend.

**Default value**

No legends

**Valid values**

Comma separated list of strings or multilingual text variables.

## **legendPos**

Optional. The position of the legend. You can add an 's' to any value if you want empty legend entries in legendText to be skipped in the legend.

### **Default value**

Legend to the right of the chart, legend entries in a vertical column.

### **Valid values**

- Right of chart, in vertical column
- Bottom of chart, in horizontal row
- Bottom of chart, in vertical column
- Top of chart, in horizontal row
- Top of chart, in vertical column
- Left of chart, in vertical column

## **legendOrder**

Optional. The order of the legend. Display in order (Default for vertical legends) shows labels in the order they were entered. Reverse order is useful in stacked bar charts to show the legend in the same order as the bars appear.

Automatic ordering (Default for horizontal legends): roughly means sorting by length, shortest first, as measured in 10 pixel blocks. When two elements are the same length (divided into 10 pixel blocks), the one listed first will appear first.

### **Default value**

Display in order

### **Valid values**

Display in order

Reverse order

Automatic ordering

**legendColor**

The color of the legend text.

**Default value**

Theme: A color compatible with the theme (if a theme is present).

**Valid values**

Color in RRGGBB hexadecimal format.

**legendFontSize**

Font size of the legend text, in points.

**Default value**

Automatically set by Google.

**Valid values**

A numeric value in points.

## **legendMargins**

Optional. Width of the margin around the legend (width, height), in pixels. Use this to avoid having the legend bump up against the chart area or the edges of the image.

### **Default value**

Automatically set by Google.

### **Valid values**

Comma separated numbers (width, height) in pixels.

## 8.3 Standard Field Visualizations

Standard field visualization webllets are shipped with LANSA. They provide a standard visualization for many of the common field types and they are used by the XSL generator (when you compile your WAM) if a specific field visualization is not specified for a field in a web\_map. For example, if you specify a field of type DATE in your web\_map, the XSL generator will use the std\_date field visualization by default when it generates the XSL for your webroutine.

A full description of the standard field visualization webllets, their properties and how to use them in your own webroutines is provided.

You should not modify the shipped webllets. Every time a Partition Initialization is executed, these webllets are re-imported and the original webllets in the repository are overwritten. If you wish to customize a field visualization webllet, save it first with a different name in the LANSA Editor and then modify the webllet copy. Don't start your own webllet names with 'std\_' as they may conflict with webllets shipped by LANSA.

<b>Webllet Name</b>	<b>Description</b>
<a href="#">Alphanumeric (std_char)</a>	A text input box control.
<a href="#">Boolean (std_boolean)</a>	A checkbox control
<a href="#">jQuery UI Datepicker (std_datepicker)</a>	A text input box with added features to support the display, entry, prompting and validation of dates.
<a href="#">jQuery UI Datetimepicker (std_datetimepicker)</a>	A text input box with added features to support the display, entry, prompting and validation of datetimes.
<a href="#">Float (std_float)</a>	A text input box control specifically configured to display and receive floating point values.
<a href="#">Input box</a>	A text input box control.

(std\_input)

Integer (std\_integer) A text input box configured to display and receive integer values with logic to ensure values entered are within the allowable range.

Object (std\_lob) A hyperlink (anchor) to the webroutine that returns the LOB content.

jQuery UI  
Timepicker  
(std\_timepicker) A text input box that supports the display, entry and validation of times

Varchar  
(std\_varchar) A text input box control.

### 8.3.1 Alphanumeric (std\_char)

#### [QuickStart - Alphanumeric Properties - Alphanumeric](#)

The alphanumeric weblet provides a text input box control. It broadly corresponds to the `<input type="text">` HTML element. The weblet looks like this (when in 'memo' mode):



The image shows a screenshot of a web browser displaying an alphanumeric weblet. At the top, there is a text input box containing the letter 'S'. Below this input box, a dropdown menu is open, listing five options: 'Sally', 'Sarah', 'Scott', and 'Sean'. The 'Scott' option is currently selected and highlighted with a light blue background. The entire control is enclosed in a thin border.

The alphanumeric weblet is used to display and receive input for character data. It is the default weblet for fixed-length character fields longer than 256.

## **QuickStart - Alphanumeric**

To use this weblet to visualize a field that is already present on your web page, open the XSL for your webroutine in the LANSAs Editor and follow the following steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Alphanumeric weblet.
2. Drag and drop the weblet over the existing field. Click on the weblet and then click on the Details tab.
3. If you dropped the weblet on an existing field, the name, value and other properties have already been set appropriately. Otherwise, set these properties now as required to associate the weblet with the required field in your webroutines web\_map.
4. If you wish the weblet to act as a multi-line input box, select 'memo' for the type property.

## Properties - Alphanumeric

The Alphanumeric weblet's properties are:

class	keyboard_shift	title
disabled	maxlength	type
display_length	name	value
display_mode	pos_absolute	width
height	read_only	word_wrap
hide_if	tab_index	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a weblet property editor. The property name is 'value' and the value is '#V1CHA02'. The value is highlighted in a blue box.

When the property loses focus, the value is shown as follows:



A screenshot of a weblet property editor. The property name is 'value' and the value is 'key('field-value', 'V1CHA02')'. The value is highlighted in a blue box.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

## **display\_length**

The approximate width of the weblet input box in characters – the browser sizes the input box according to the number of characters specified. If the width property is specified, it takes precedence and the display\_length property is ignored.

### **Default value**

Blank (the weblet assumes a default size).

### **Valid values**

A numeric value.

## **type**

Specifies the type of input control the weblet implements. This weblet is designed to implement an `<input type=text>` or an `<input type=password>` control or a `<textarea>` control. Other types are possible but are not supported for this weblet.

## **Default value**

'text'

## **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

- 'text'**        Creates a text entry control.
- 'memo'**        Creates a multi-line text entry control with word-wrapping.
- 'password'**    Creates a text entry control in which characters output or typed are not visible – instead an asterisk or other placeholder character is shown.

## **keyboard\_shift**

The keyboard shift for the input field.

### **Default value**

The keyboard shift of the field with this weblet visualization. Blank otherwise.

### **Valid values**

' ', 'W', 'J', 'E', 'O' and 'U'

The keyboard shift is currently only used to validate DBCS fields.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

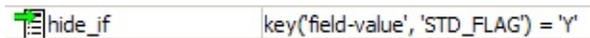
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is '#STD\_FLAG = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **word\_wrap**

If 'memo' is specified for the type property, this property specifies how the weblet should handle word-wrapping when typing text.

### **Default value**

Blank.

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

- 'soft'** Text is displayed with word wrapping and submitted without carriage returns and line feeds.
- 'hard'** Text is displayed with word wrapping and submitted with soft returns and line feeds.
- 'off'** Word wrapping is disabled. The lines appear exactly as the user types them.

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

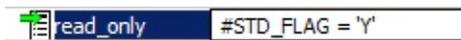
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSA Editor. This resulted in the value shown for the pos\_absolute property.

 pos\_absolute 'position:absolute;left: 312.768pt; top: 192.024pt;'

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.3.2 Boolean (std\_boolean)

### [QuickStart - Boolean Properties - Boolean](#)

The boolean weblet provides a checkbox control. It broadly corresponds to the `<input type="checkbox">` HTML element. It extends the checkbox weblet with particular support for boolean type fields.

A checkbox control is typically used to represent a value that can have one of two states. For the Boolean weblet the checkbox represents the true and false states.

When used in a list, the boolean weblet looks like this:

<input type="checkbox"/>	Dept Code	Department Description
<input type="checkbox"/>	ADM	ADMINISTRATOR DEPT
<input checked="" type="checkbox"/>	AUD	INTERNAL AUDITING
<input type="checkbox"/>	FLT	FLEET ADMINISTRATION

## QuickStart - Boolean

Because the boolean weblet is the default visualization for boolean fields you usually do not need to manually add it to your web page. Simply include your boolean fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the Boolean weblet.

If you need to add the weblet to your page manually, simply drag the boolean field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Boolean weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.

## Properties - Boolean

The Boolean weblet's properties are:

class            mouseover\_class   tab\_index  
display\_mode   name                value  
hide\_if           pos\_absolute

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the checkbox and/or that is used to receive the state of the checkbox.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

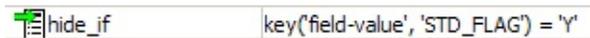
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is '#STD\_FLAG = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The 'hide\_if' property is selected. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**mouseover\_class**

The Cascading Style Sheet (CSS) class name of the weblet when the mouse is moved over it.

**Default value**

No default value applies for this weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

### 8.3.3 jQuery UI Datepicker (std\_datepicker)

#### [QuickStart - Datepicker Properties - Datepicker](#)

The datepicker weblet provides a text input box control with added features to support the display, entry, prompting and validation of dates. It broadly corresponds to the `<input type="text">` HTML element.

An example of the weblet is shown below. This example shows the datepicker with default settings:



The datepicker weblet is designed to work with the date data type, which is stored in UTC format.

## QuickStart - Datepicker

Because the datepicker weblet is the default visualization for fields of type date, you usually do not need to manually add it to your web page. Simply include your date fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the datepicker weblet. Similarly fields of type time and of type datetime will be visualized using the timepicker (std\_timepicker) and datetimepicker (std\_datetimepicker) weblets.

**Date fields** created before Version 12 SP1 use std\_date weblet by default. To use std\_datepicker, change the weblet visualization in the field definition.

The Datepicker weblet uses the ISO language code to localize some properties such as date format, first day of the week, and calendar captions.

If you do need to add the datepicker weblet to your page manually, simply drag the date field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the jQuery UI Datepicker weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.

## Displaying Dates Stored in Numeric Fields

Use intrinsics to map dates stored in numeric fields to date fields:

\* Date stored in MMDDYY format in numeric field

```
Webroutine Name(TO_DATE) Desc('Converting from number to date')  
  Web_Map For(*output) Fields(#dat01)  
  #std_date := 061525  
  #dat01 := #std_date.AsDate(MMDDYY)  
Endroutine
```

```
Webroutine Name(FROM_DATE) Desc('Converting from date to number')  
  Web_Map For(*input) Fields(#dat01)  
  #std_date := #dat01.AsNumber(MMDDYY)  
Endroutine
```

## Properties - Datepicker

The Datepicker weblet's properties are:

allow_sqlnull	firstDay	showInline
autoSize	hide_if	showMonthAfterYear
buttonImage	maxDate	showOn
buttonText	minDate	showOtherMonths
changeMonth	name	tab_index
changeYear	onchange_script	title
dateFormat	pos_absolute	value
disabled	selectOtherMonths	width
display_mode	shortYearCutoff	yearRange
duration	showAnim	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **allow\_sqlnull**

A Boolean property which determines if the date value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## **dateFormat**

The input format of the date. The default 'Auto' uses the default for the language (regional setting).

See the Datetimepicker weblet for a full list of valid format specifiers.

**Note:** this specifies the presentation format the weblet uses. The input and output date received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

### **Default value**

'Auto'. Uses the default for the language (regional setting).

### **Valid values**

Any of the values listed in the property dropdown.

**firstDay**

Sets the first day of the week.

**Default value**

Auto. Uses the default regional setting.

**Valid values**

A valid day of the week. Select from the property dropdown.

## changeMonth

If true, you can change the month by selecting from a drop-down list.

## Default value

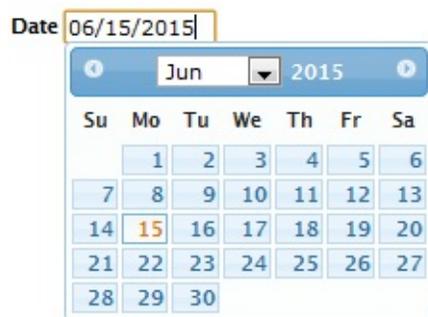
False

## Valid values

true(), false() or a valid expression.

## Example

Datepicker with property set to true().



## changeYear

If true, you can change the year by selecting from a drop-down list.

## Default value

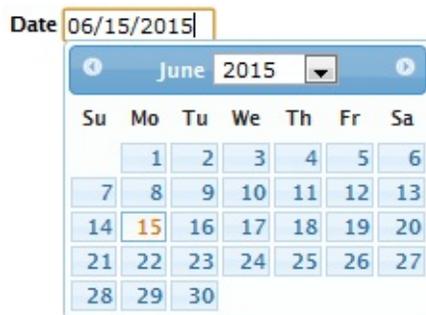
False

## Valid values

true(), false() or a valid expression.

## Example

Datepicker with property set to true().



## **yearRange**

The range of years displayed in the year drop-down: either relative to today's year (-nn:+nn), relative to the currently selected year (c-nn:c+nn), absolute (nnnn:nnnn), or combinations of these formats (nnnn:-nn). Note that this option only affects what appears in the drop-down, to restrict which dates may be selected use the minDate and/or maxDate options.

### **Default value**

+/1 10 selected year

### **Valid values**

c-nn:c+nn: Where nn is the number of years (range relative to selected year)

-nn:+nn: Range relative to current year.

nnnn:nnnn: Absolute years.

### **Example**

c-2:c+2: Range of two years around selected year

-1:+3: From one year ago to 3 years ahead

2000:2050: Range from years 2000 to 2050

## showOtherMonths

If true, displays dates in other months (non-selectable) at the start or end of the current month. To make these days selectable use selectOtherMonths.

### Default value

False

### Valid values

true(), false() or a valid expression.

### Example

Datepicker with property set to true().



## **selectOtherMonths**

If true, days in other months shown before or after the current month are selectable. This only applies if showOtherMonths is also true.

### **Default value**

False

### **Valid values**

true(), false() or a valid expression.

## **minDate**

Set a minimum selectable date via a string in the current dateFormat, or a number of days from today (e.g. '-7') or a string of values and periods ('y' for years, 'm' for months, 'w' for weeks, 'd' for days, e.g. '-1y -1m'), or null for no limit.

### **Default value**

no limit

### **Valid values**

A valid expression (as per in the description) or select any of the predefined values from the property dropdown.

### **Example**

-1y: -1 year.

## **maxDate**

Set a maximum selectable date via a string in the current dateFormat, or a number of days from today (e.g. '+7') or a string of values and periods ('y' for years, 'm' for months, 'w' for weeks, 'd' for days, e.g. '+1m +1w'), or null for no limit.

### **Default value**

no limit

### **Valid values**

A valid expression (as per in the description) or select any of the predefined values from the property dropdown.

### **Example**

+1w: +1 week.

## **shortYearCutoff**

Set the cutoff year for determining the century for a date (used in conjunction with dateFormat 'y'). If a numeric string only ('0'-'99') is provided then this value is used directly. If a string value has a '+' then it is added to the current year. Once the cutoff year is calculated, any dates entered with a year value less than or equal to it are considered to be in the current century, while those greater than it are deemed to be in the previous century.

### **Default value**

+10

### **Valid values**

A string representing an cutoff year

### **Example**

+20: If the current year is 2015, years 00 to 35 are considered to be years 2000 to 2035. 36 to 99 are considered to be 1936 to 1999.

**showInline**

If true, Display the datepicker embedded in the page instead of in an overlay.

**Default value**

False

**Valid values**

true(), false() or a valid expression.

## **showOn**

Whether the datepicker appear automatically when the field receives focus, appear only when a button is clicked, or appear when either event takes place.

### **Default value**

focus

### **Valid values**

focus, button or both

### **Example**

Show on button click:

Date  

## **showMonthAfterYear**

If true, the month is placed after the year in the header. This attribute is one of the regionalization attributes. The default 'Auto' uses the default for the language (regional setting).

### **Default value**

Auto

### **Valid values**

true(), false() or a valid expression.

## **buttonImage**

The path and file name, relative to the images virtual directory, of the image to display on the calendar prompt button.

### **Default value**

'calendar\_jqui.gif' (this image is shipped with LANSA).

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**buttonText**

The text that may appear as tip text on mouse hover over the datepicker button.

**Default value**

Blank – the text specified for the title property is used.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page. The weblet will reserve a minimum width based on the data to be displayed.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

**autoSize**

Set to true to automatically resize the input field to accomodate dates in the current date format.

**Default value**

False

**Valid values**

true(), false() or a valid expression.

**showAnim**

Sets the name of the animation used to show/hide the datepicker.

**Default value**

show

**Valid values**

show, slideDown or fadeIn

**duration**

Controls the speed at which the datepicker appears. Choose one of three predefined speeds.

**Default value**

normal

**Valid values**

slow, normal or fast

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

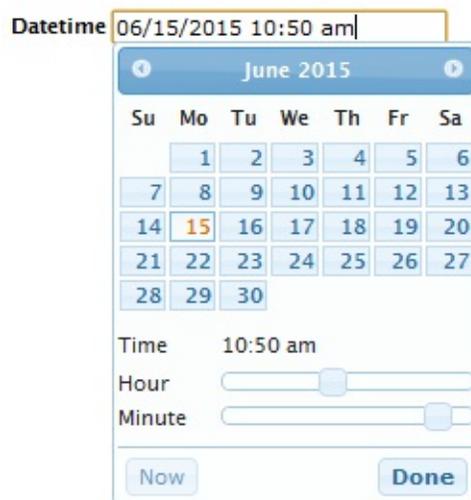
Any valid JavaScript statement(s).

## 8.3.4 jQuery UI Datetimepicker (std\_datetimepicker)

### QuickStart - Datetimepicker Properties - Datetimepicker

The datetimepicker widget provides a text input box control with added features to support the display, entry, prompting and validation of datetimes. It broadly corresponds to the `<input type="text">` HTML element.

An example of the widget is shown below. This example shows the datetimepicker with default settings:



The datetimepicker widget is designed to work with the datetime data type, which is stored in UTC format.

## QuickStart - Datetimestpicker

Because the datetimestpicker weblet is the default visualization for fields of type datetime, you usually do not need to manually add it to your web page. Simply include your date fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the datetimestpicker weblet. Similarly fields of type time and of type date will be visualized using the timepicker (std\_timepicker) and datepicker (std\_datepicker) weblets.

Datetime fields created before version 12 SP1 use the std\_datetime weblet by default. To use std\_datetimestpicker, change the weblet visualization in the field definition.

The Datetimestpicker weblet uses the ISO language code to localize some properties (e.g. date format, first day of the week) and calendar and time slider captions.

If you do need to add the datetimestpicker weblet to your page manually, simply drag the datetime field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the jQuery UI Datetimestpicker weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.

## Properties - Datetimepicker

The Datetimepicker weblet's properties are:

allow_sqlnull	hide_if	showAnim
autoSize	hourMax	showMonthAfterYear
buttonImage	hourMin	showOn
buttonText	maxDate	showOtherMonths
changeMonth	minDate	stepHour
changeYear	minuteMax	stepMinute
dateFormat	minuteMin	stepSecond
disabled	name	tab_index
display_in_utc	onchange_script	timeFormat
display_mode	pos_absolute	title
duration	selectOtherMonths	value
firstDay	shortYearCutoff	width
		yearRange

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **display\_in\_utc**

A Boolean property which determines if the datetimepicker displays the datetime's UTC value or the datetime's local value.

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the DUTC attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## **allow\_sqlnull**

A Boolean property which determines if the datetime value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## dateFormat

The input format of the date portion of the datetime. The default 'Auto' uses the default for the language (regional setting).

**Note:** this specifies the presentation format the weblet uses. The input and output datetime received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

### Default value

'Auto'. Uses the default for the language (regional setting).

### Valid values

The following formats are supported:

<b>Date Format</b>	<b>Example</b>
dd/mm/yyyy	09/06/2025
dd/mm/yy	09/06/25
dd-mm-yyyy	09-06-2025
dd-mm-yy	09-06-25
mm/dd/yyyy	06/09/2025
mm/dd/yy	06/09/25
mm-dd-yyyy	06-09-2025
mm-dd-yy	06-09-25
yyyy-mm-dd	2025-06-09
yy-mm-dd	25-06-09
yyyy/mm/dd	2025/06/09
yy/mm/dd	25/06/09
d-M-yy	9-Jun-25

d-M-yyyy	9-Jun-2025
d M, yy	9 Jun, 25
d M, yyyy	9 Jun, 2025
d-MM-yy	9-Jun-25
d-MM-yyyy	9-Jun-2025
d MM, yy	9 Jun, 25
d MM, yyyy	9 Jun, 2025
dd-M-yy	09-Jun-25
dd-M-yyyy	09-Jun-2025
dd M, yy	09 Jun, 25
dd M, yyyy	09 Jun, 2025
dd-MM-yy	09-June-25
dd-MM-yyyy	09-June-2025
dd MM, yy	09 June, 25
dd MM, yyyy	09 June, 2025
yy-M-d	25-Jun-9
yyyy-M-d	2025-Jun-9
yy M, dd	25 Jun, 09
yyyy M, dd	2025 Jun, 09
yy-MM-d	25-June-9
yyyy-MM-d	2025-June-9
yy MM, d	25 June, 9
yyyy MM, d	2025 June, 9
yy-M-dd	25-Jun-09
yyyy-M-dd	2025-Jun-09

yy M, dd	25 Jun, 09
yyyy M, dd	2025 Jun, 09
yy-MM-dd	25-June-09
yyyy-MM-dd	2025-June-09
yy MM, dd	25 June, 09
yyyy MM, dd	2025 June, 09
DDDDDD, d MM yyyy	Monday, 9 June 2025
DDDDDD, d MM yy	Monday, 9 June 25
DDDDDD, dd MM yyyy	Monday, 09 June 2025
DDDDDD, dd MM yy	Monday, 09 June 25

## **timeFormat**

The input format of the time portion of the datetime. The default 'Auto' uses the default for the language (regional setting).

### **Default value**

'Auto'. Uses the default for the language (regional setting).

### **Valid values**

The following formats are supported:

<b>Time Format</b>	<b>Example</b>
H:mm	15:30
H:mm:ss	15:30:00
HH:mm	15:30
HH:mm:ss	15:30:00
h:mm T	3:30 PM
h:mm t	3:30 pm
hh:mm T	03:30 PM
hh:mm t	03:30 pm
hh:mm:ss T	03:30:00 PM
hh:mm:ss t	03:30:00 pm

**firstDay**

Sets the first day of the week.

**Default value**

Auto. Uses the default regional setting.

**Valid values**

A valid day of the week. Select from the property dropdown.

## changeMonth

If true, you can change the month by selecting from a drop-down list.

## Default value

False

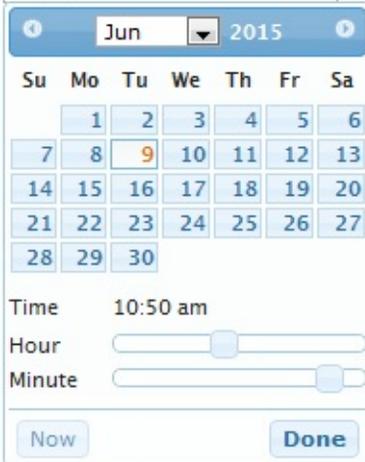
## Valid values

true(), false() or a valid expression.

## Example

Datetimepicker with property set to true():

Datetime 06/09/2015 10:50 am



The image shows a datetime picker interface. At the top, there is a text input field containing "06/09/2015 10:50 am". Below this is a calendar view for June 2015. The calendar has a header with "Jun" and "2015" and a dropdown arrow. The days of the week are labeled "Su", "Mo", "Tu", "We", "Th", "Fr", "Sa". The dates are arranged in a grid: 1-6 in the first row, 7-13 in the second, 14-20 in the third, 21-27 in the fourth, and 28-30 in the fifth. The date "9" is highlighted in orange. Below the calendar, there is a "Time" section showing "10:50 am". Underneath, there are two sliders: "Hour" and "Minute". At the bottom, there are two buttons: "Now" and "Done".

## changeYear

If true, you can change the year by selecting from a drop-down list.

## Default value

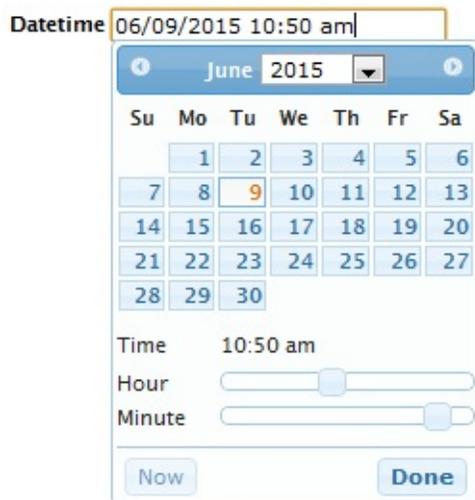
False

## Valid values

true(), false() or a valid expression.

## Example

Datetimepicker with property set to true():



## **yearRange**

The range of years displayed in the year drop-down: either relative to today's year (-nn:+nn), relative to the currently selected year (c-nn:c+nn), absolute (nnnn:nnnn), or combinations of these formats (nnnn:-nn). Note that this option only affects what appears in the drop-down, to restrict which dates may be selected use the minDate and/or maxDate options.

### **Default value**

+/1 10 selected year

### **Valid values**

c-nn:c+nn: Where nn is the number of years (range relative to selected year)

-nn:+nn: Range relative to current year.

nnnn:nnnn: Absolute years.

### **Example**

c-2:c+2: Range of two years around selected year

-1:+3: From one year ago to 3 years ahead

2000:2050: Range from years 2000 to 2050

## showOtherMonths

If true, displays dates in other months (non-selectable) at the start or end of the current month. To make these days selectable use selectOtherMonths.

### Default value

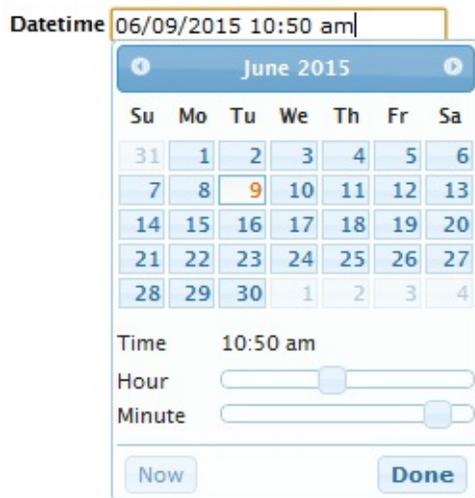
False

### Valid values

true(), false() or a valid expression.

### Example

Datetimepicker with property set to true():



## **selectOtherMonths**

If true, days in other months shown before or after the current month are selectable. This only applies if showOtherMonths is also true.

### **Default value**

False

### **Valid values**

true(), false() or a valid expression.

## **minDate**

Set a minimum selectable date via a string in the current dateFormat, or a number of days from today (e.g. '-7') or a string of values and periods ('y' for years, 'm' for months, 'w' for weeks, 'd' for days, e.g. '-1y -1m'), or null for no limit.

### **Default value**

no limit

### **Valid values**

A valid expression (as per in the description) or select any of the predefined values from the property dropdown.

### **Example**

-1y: -1 year.

## **maxDate**

Set a maximum selectable date via a string in the current dateFormat, or a number of days from today (e.g. '+7') or a string of values and periods ('y' for years, 'm' for months, 'w' for weeks, 'd' for days, e.g. '+1m +1w'), or null for no limit.

### **Default value**

no limit

### **Valid values**

A valid expression (as per in the description) or select any of the predefined values from the property dropdown.

### **Example**

+1w: +1 week.

**stepHour**

Hours step interval in the datetimepicker hour slider

**Default value**

1 hour

**Valid values**

An integer

**Example**

1 (1 hour)

**stepMinute**

Minutes step interval in the datetimepicker minute slider

**Default value**

1 minute

**Valid values**

An integer

**Example**

15 (15 minutes)

**stepSecond**

Seconds step interval in the datetimepicker seconds slider

**Default value**

1 second

**Valid values**

An integer

**Example**

10 (10 seconds)

## **shortYearCutoff**

Set the cutoff year for determining the century for a date (used in conjunction with dateFormat 'y'). If a numeric string only ('0'-'99') is provided then this value is used directly. If a string value has a '+' then it is added to the current year. Once the cutoff year is calculated, any dates entered with a year value less than or equal to it are considered to be in the current century, while those greater than it are deemed to be in the previous century.

### **Default value**

+10

### **Valid values**

A string representing an cutoff year

### **Example**

+20: If the current year is 2015, years 00 to 35 are considered to be years 2000 to 2035. 36 to 99 are considered to be 1936 to 1999.

**hourMin**

Minimum hour in Timepicker slider

**Default value**

0

**Valid values**

0 to 23

**hourMax**

Maximum hour in Timepicker slider

**Default value**

23

**Valid values**

0 to 23

**minuteMin**

Minimum minute in Timepicker slider

**Default value**

0

**Valid values**

0 to 59

**minuteMax**

Maximum minute in Timepicker slider

**Default value**

59

**Valid values**

0 to 59

## **showOn**

Whether the datetimepicker appear automatically when the field receives focus, appear only when a button is clicked, or appear when either event takes place.

## **Default value**

focus

## **Valid values**

focus, button or both

## **Example**

Show on button click:

Datetime  

## **showMonthAfterYear**

If true, the month is placed after the year in the header. This attribute is one of the regionalization attributes. The default 'Auto' uses the default for the language (regional setting).

### **Default value**

Auto

### **Valid values**

true(), false() or a valid expression.

## **buttonImage**

The path and file name, relative to the images virtual directory, of the image to display on the calendar prompt button.

### **Default value**

'calendar\_jqui.gif' (this image is shipped with LANSA).

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**buttonText**

The text that may appear as tip text on mouse hover over the datetimepicker button.

**Default value**

Blank – the text specified for the title property is used.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page. The weblet will reserve a minimum width based on the data to be displayed.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

**autoSize**

Set to true to automatically resize the input field to accomodate datetimes in the current date and time format.

**Default value**

False

**Valid values**

true(), false() or a valid expression.

## **showAnim**

Sets the name of the animation used to show/hide the datetimepicker.

### **Default value**

show

### **Valid values**

show, slideDown or fadeIn

**duration**

Controls the speed at which the datetimestepicker appears. Choose one of three predefined speeds.

**Default value**

normal

**Valid values**

slow, normal or fast

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

### 8.3.5 Float (std\_float)

#### [QuickStart - Float Properties - Float](#)

The float weblet provides a text input box control specifically configured to display and receive floating point values and includes validation logic to ensure that values entered are within the allowable range for the specified floating point size. The weblet looks like this (for clarity, the label for the input box is shown but is not part of the weblet):

**Float 4**

The float weblet is the default visualization for floating point fields. It broadly corresponds to the `<input type="text">` HTML element.

## **QuickStart - Float**

Because the float weblet is the default visualization for floating point fields you usually do not need to manually add it to your web page – instead, simply customize the weblet properties as required.

If you do need to add the weblet to your page manually, simply drag the floating point field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSA Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Float weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.
4. Set the maxlength and size properties according to the field definition.

## Properties - Float

The Float weblet's properties are:

class	maxlength	tab_index
disabled	name	title
display_mode	pos_absolute	type
height	read_only	value
hide_if	size	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

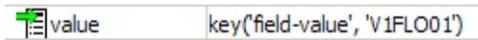
## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The property name is 'value' and its value is '#V1FLO01'. The editor has a blue header bar with the property name and a white input field with the value.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The property name is 'value' and its value is 'key('field-value', 'V1FLO01')'. The editor has a white header bar with the property name and a white input field with the value.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field, including allowances for a sign and numeric editing.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

**size**

The size of the weblet data in characters/bytes. For the float weblet this is used to validate the allowable range of values that can be entered.

**Default value**

Blank (the weblet assumes the maximum floating point size for validation).

**Valid values**

Floating point fields may be 4 or 8 bytes in length. One of those values should be used for correct validation.

## **type**

Specifies the type of input control the weblet implements. This weblet is designed to implement an `<input type=text>` or an `<input type=password>` control. Other types are possible but are not supported for this weblet.

## **Default value**

'text'

## **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

'text'      Creates a text entry control.

'password'      Creates a text entry control in which characters output or typed are not visible – instead an asterisk or other placeholder character is shown.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

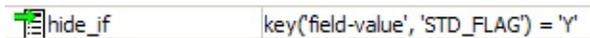
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is '#STD\_FLAG = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. A small icon of a document with a green checkmark is visible to the left of the property name.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

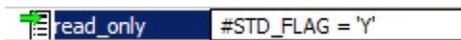
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor for the 'read\_only' property. The property name 'read\_only' is on the left, and the expression '#STD\_FLAG = 'Y'' is entered in the text field on the right.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor for the 'read\_only' property. The property name 'read\_only' is on the left, and the expression 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in the text field on the right.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.3.6 Input box (std\_input)

[QuickStart - Input box](#) [Properties - Input box](#)

The input box weblet provides a text input box control. It broadly corresponds to the `<input type="text">` HTML element. The weblet looks like this (for clarity, the label for the input box is shown but is not part of the weblet):

**Department Code**

The input box is used to display and receive input for both numeric and character data. It is a generalized weblet upon which a number of other more specialized weblets are based. In many cases one of the specialized weblets may better suit your purpose. They include `std_char`, `std_varchar`, `std_float` and `std_integer`.

## QuickStart - Input box

To use this weblet to visualize a field that is already present on your web page, open the XSL for your webroutine in the LANSA Editor and follow the following steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Input box weblet.
2. Drag and drop the weblet over the existing field. Click on the weblet and then click on the Details tab.
3. If you dropped the weblet on an existing field, the name, value and other properties have already been set appropriately. Otherwise, set these properties now as required to associate the weblet with the required field in your webroutines web\_map.

**Note:** depending on the definition of the field in your webroutine, the generated XSL may already visualize the field using one of the std\_char or std\_varchar field visualization weblets. These weblets provide much of the functionality of the input box weblet with additional features.

## Properties - Input box

The Input box weblet's properties are:

class	keyboard_shift	size
disabled	maxlength	tab_index
display_length	name	title
display_mode	onchange_script	type
height	pos_absolute	value
hide_if	read_only	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The 'value' property is highlighted with a blue background. The value is set to '#DEPARTMENT'. To the right of the value, the text '#DEPARTMENT' is visible.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The 'value' property is highlighted with a blue background. The value is set to 'key("field-value", 'DEPARTMENT')'. To the right of the value, the text 'key("field-value", 'DEPARTMENT')' is visible.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

**size**

The size of the weblet data in characters/bytes.

This property is currently not implemented – use the `maxlength` and/or `display_length` properties instead.

## **display\_length**

The approximate size of the weblet input box in characters – the browser sizes the input box according to the number of characters specified. If the width property is specified, it takes precedence and the display\_length property is ignored.

### **Default value**

Blank (the weblet assumes a default size).

### **Valid values**

A numeric value.

## **type**

Specifies the type of input control the weblet implements. This weblet is designed to implement an `<input type=text>` or an `<input type=password>` control. Other types are possible but are not supported for this weblet.

## **Default value**

'text'

## **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

'text'      Creates a text entry control.

'password'      Creates a text entry control in which characters output or typed are not visible – instead an asterisk or other placeholder character is shown.

## **keyboard\_shift**

The keyboard shift for the input field.

### **Default value**

The keyboard shift of the field with this weblet visualization. Blank otherwise.

### **Valid values**

Char and String data types: ' ', 'W', 'J', 'E', 'O' and 'U'

Alpha data type: ' ', 'X', 'A', 'N', "W", 'T', 'D', 'M', 'J', 'E' and 'O'

The keyboard shift is currently only used to validate DBCS fields.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

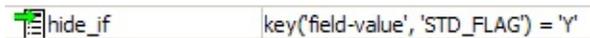
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value '#STD\_FLAG = 'Y'' is entered in a text field.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in a text field.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

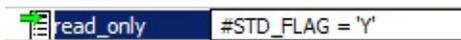
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''. The dropdown menu has a small icon to its left.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''. The dropdown menu has a small icon to its left.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

### 8.3.7 Integer (std\_integer)

#### [QuickStart - Integer Properties - Integer](#)

The integer weblet provides a text input box control specifically configured to display and receive integer values and includes validation logic to ensure that values entered are within the allowable range for the specified integer size. The weblet looks like this (for clarity, the label for the input box is shown but is not part of the weblet):

**Integer 4**

The integer weblet is the default visualization for integer fields. It broadly corresponds to the `<input type="text">` HTML element.

## QuickStart - Integer

Because the integer weblet is the default visualization for integer fields you usually do not need to manually add it to your web page – instead, simply customize the weblet properties as required.

If you do need to add the weblet to your page manually, simply drag the integer field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Integer weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.
4. Set the maxlength and size properties according to the field definition.

## Properties - Integer

The Integer weblet's properties are:

class	maxlength	tab_index
disabled	name	title
display_mode	pos_absolute	type
height	read_only	value
hide_if	size	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

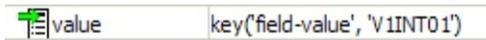
## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The 'value' property is highlighted with a blue background. The value is set to `#V1INT01`.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The 'value' property is highlighted with a blue background. The value is set to `key('field-value', 'V1INT01')`.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field, including allowances for a sign and numeric editing.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

**size**

The size of the weblet data in characters/bytes. For the integer weblet this is used to validate the allowable range of values that can be entered.

**Default value**

Blank (the weblet assumes the maximum integer size for validation).

**Valid values**

Integers may be 1, 2, 4 or 8 bytes in length. One of those values should be used for correct validation.

## **type**

Specifies the type of input control the weblet implements. This weblet is designed to implement an `<input type=text>` or an `<input type=password>` control. Other types are possible but are not supported for this weblet.

## **Default value**

'text'

## **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

**'text'**          Creates a text entry control

**'password'**    Creates a text entry control in which characters output or typed are not visible – instead an asterisk or other placeholder character is shown.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

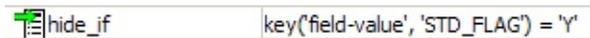
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The 'hide\_if' property is selected and highlighted in blue. The value entered is '#STD\_FLAG = 'Y''. To the left of the text is a small icon representing a document with a green checkmark.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The 'hide\_if' property is selected. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. To the left of the text is a small icon representing a document with a green checkmark.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

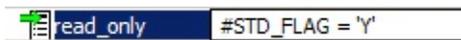
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''. The dropdown menu has a small icon to its left, and the text input field has a small icon to its left.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''. The dropdown menu has a small icon to its left, and the text input field has a small icon to its left.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

### 8.3.8 Object (std\_lob)

#### QuickStart - Object Properties - Object

The Object weblet provides a hyperlink control. It broadly corresponds to the <a> (anchor) HTML element that designates the destination of a hypertext link. It is very similar in function to the standard Anchor weblet (std\_anchor).

- The Object weblet can display an image and/or text to represent the link. It is used to specify the WAM and webroutine that serves the Object content.
- The image or text can be static (specified as literals in the weblet properties) or can be determined by nominated fields, system variables or multilingual variables.

Note: "Object" in the context of this weblet refers to the contents of a LOB or a stream file.

The Object weblet looks like this:

[Sample Document](#)

In the example above, when the user clicks on the hyperlink the webroutine that serves the Object is invoked and the Object is sent to the user agent.

## QuickStart - Object

To use a Object follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Object weblet.
2. Drag the Object weblet over an empty space or over a LOB field and release the left-mouse button. Select the hyperlink and then click on the Details tab If you dropped the weblet on a LOB field, you should see that the name and value properties for the Object weblet have already been set according to the field upon which it was dropped.
3. If the weblet is used in a list column, set the currentrowhfield and currentrownumval properties as described in the property descriptions (if required).
4. Set the on\_click\_wrname property to the name of the webroutine that serves the Object. If the webroutine is in a different WAM to the current webroutine then you will need to set the on\_click\_wamname property as well.

## Properties - Object

The Object weblet's properties are:

absolute-image-path	on_click_wamname	show_in_new_window
class	on_click_wname	tab_index
currentrowhfield	pos_absolute_design	target_window_name
currentrownumval	presubmit_js	text_class
formname	protocol	value
hide_if	reentryfield	vf_wamevent
mouseover_class	reentryvalue	width_design
name	relative-image-path	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

This property specifies the text that is displayed for the hyperlink. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies if not dropped over a field. If dropped over a field, the default is the value of the field.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list). The hyperlink may also display an image in addition to or instead of text – see the `relative-image-path` and `absolute-image-path` properties.

## **currentrowhfield**

The field name to be used to post to the target webroutine the value that is specified in the currentrownumval property. The field name should be in single quotes.

See the description of the currentrownumval property for further information.

### **Default value**

'STDROWNUM'

### **Valid values**

Single-quoted text.

### **Example**

This example specifies the field name ROWNUM as the field name to be used to post the value to the target webroutine. The target webroutine would need to have field ROWNUM in its WEB\_MAP for \*BOTH or for \*INPUT in order to receive the value:

 currentrowhfield	'DEPTLINK'
--	------------

## currentrownumval

The value to post to the target webroutine in the field specified in the currentrowhfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the currentrowhfield property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. currentrowhfield: the field name that the target webroutine uses to refer to the information
2. currentrownumval: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** Despite the name of the property being *currentrownumval*, the field name specified in currentrownumval is not required to be a numeric field.

### Default value

position()

### Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

### Example

This example specifies the ROWNUM field from the current list row as the field whose value is posted to the target webroutine:

```
r[ currentrowhfield ROWNUM
```

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

See the description of the [reentryvalue](#) property for further information.

### **Default value**

'STDREENTRY'

### **Valid values**

Single-quoted text.

## **reentryvalue**

The value to post to the target webroutine in the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the reentryfield property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. **reentryfield**: the field name that the target webroutine uses to refer to the information
2. **reentryvalue**: a literal value or a field name in this (the source) webroutine that contains the necessary information

## **Default value**

'D'

## **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

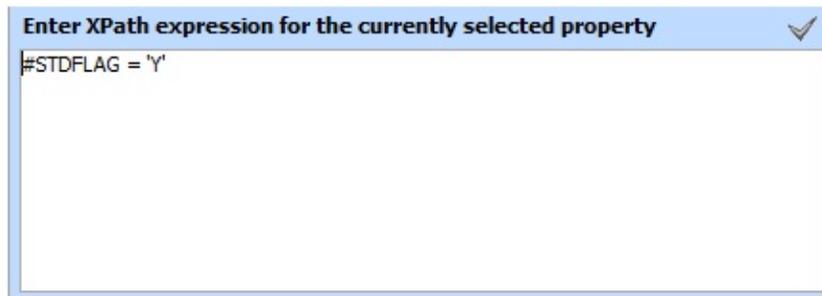
False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in the XPath expression area:



A screenshot of a dialog box titled "Enter XPath expression for the currently selected property" with a checkmark icon in the top right corner. The dialog contains a text input field with the text "#STDFLAG = 'Y'" entered.

When the property loses focus, the expression is shown as follows:

 hide\_if      key('field-value', 'STDFLAG') = 'Y'

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

**on\_click\_wamname**

Specifies the name of the WAM whose webroutine sends the Object when the hyperlink is clicked. (The webroutine name is specified in the on\_click\_wname property.)

**Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName)

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **on\_click\_wrname**

Specifies the name of the webroutine that sends the Object when the hyperlink is clicked. (The name of the WAM containing the webroutine is specified in the on\_click\_wamname property.)

### **Default value**

No default value applies – the on\_click\_wrname property must be specified

### **Valid values**

The name of an Object serving Webroutine in single quotes. The Object serving Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

Typically you might use this property when it is necessary to switch to or from secure-mode processing. Otherwise it is not usually necessary to specify this property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. If specified, it is usually 'http:' or 'https:'.

## **show\_in\_new\_window**

A boolean property, the result of which determines whether the Object should be shown in a new browser window.

### **Default value**

false() – response Object is shown in the current browser window.

### **Valid values**

true(), false() or a valid expression.

**target\_window\_name**

The name of the window, or frame, in which the Object will be shown.

**Default value**

Blank – the Object will be shown in the current window.

**Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that Position Absolutely must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned)

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width\_design**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design property. However you can directly edit the property value if required.

### **Default value**

Blank (weblet uses its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **relative-image-path**

The path and file name, relative to the images virtual directory, of the image to be displayed.

### **Default value**

Blank – no image is displayed.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **absolute-image-path**

The path and file name of the image to be displayed. If specified, the `relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**mouseover\_class**

The Cascading Style Sheet (CSS) class name of the weblet when the mouse is moved over it.

**Default value**

No default value applies for this weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**text\_class**

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

The name of the shipped text class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **presubmit\_js**

JavaScript code to be run prior to navigating to the destination of the hyperlink.  
JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**vf\_wamevent**

VLf WAM event string

**Default value**

Blank.

**Valid values**

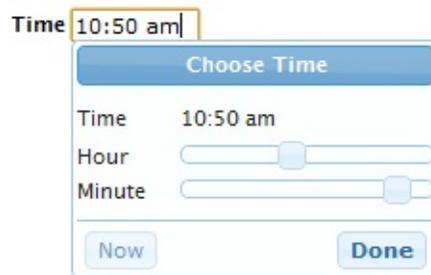
String value. Comma (',') not allowed.

### 8.3.9 jQuery UI Timepicker (std\_timepicker)

#### [QuickStart - Timepicker Properties - Timepicker](#)

The timepicker weblet provides a text input box control with added features to support the display, entry and validation of times. It broadly corresponds to the `<input type="text">` HTML element.

An example of the weblet is shown below:



The timepicker weblet is used with fields of type time. If you use this type, the data will automatically be passed in the format expected by the weblet.

## QuickStart - Timepicker

Because the timepicker weblet is the default visualization for fields of type time, you usually do not need to manually add it to your web page. Simply include your time fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the time weblet. Similarly fields of type date and of type datetime will be visualized using the datepicker (std\_datepicker) and datetimepicker (std\_datetimepicker) weblets.

Time fields created before version 12 SP1 use by default std\_time weblet. To use std\_timepicker, change the weblet visualization in the field definition.

The Timepicker weblet uses the ISO language code to localize the time format and time slider captions.

If you do need to add the timepicker weblet to your page manually, simply drag the time field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSA Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Timepicker weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.

## Properties - Timepicker

The Timepicker weblet's properties are:

allow_sqlnull	hourMin	showOn
autoSize	hourMax	stepHour
buttonImage	minuteMin	stepMinute
buttonText	minuteMax	stepSecond
disabled	name	tab_index
display_mode	onchange_script	timeFormat
duration	pos_absolute	title
hide_if	read_only	value
	showAnim	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

**Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **allow\_sqlnull**

A Boolean property which determines if the timepicker value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

**timeFormat**

The input format of the timepicker. The default 'Auto' uses the default for the language (regional setting).

**Default value**

'Auto'. Uses the default for the language (regional setting).

**Valid values**

See the Datetimepicker weblet for a full list of valid format specifiers.

**hourMin**

Minimum hour in Timepicker slider

**Default value**

0

**Valid values**

0 to 23

**hourMax**

Maximum hour in Timepicker slider

**Default value**

23

**Valid values**

0 to 23

**minuteMin**

Minimum minute in Timepicker slider

**Default value**

0

**Valid values**

0 to 59

**minuteMax**

Maximum minute in Timepicker slider

**Default value**

59

**Valid values**

0 to 59

**stepHour**

Hours step interval in the timepicker hour slider

**Default value**

1 hour

**Valid values**

An integer

**Example**

1 (1 hour)

**stepMinute**

Minutes step interval in the timepicker minute slider

**Default value**

1 minute

**Valid values**

An integer

**Example**

15 (15 minutes)

**stepSecond**

Seconds step interval in the timepicker seconds slider

**Default value**

1 second

**Valid values**

An integer

**Example**

10 (10 seconds)

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **showOn**

Whether the timepicker appear automatically when the field receives focus, appear only when a button is clicked, or appear when either event takes place.

### **Default value**

focus

### **Valid values**

focus, button or both

### **Example**

Show on button click:

Time  

## **buttonImage**

The path and file name, relative to the images virtual directory, of the image to display on the timepicker prompt button.

### **Default value**

'calendar\_jqui.gif' (this image is shipped with LANSA).

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

**buttonText**

The text that may appear as tip text on mouse hover over the timepicker button.

**Default value**

Blank – the text specified for the title property is used.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**read\_only**

A Boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

**Default value**

Blank – equivalent to False (that is, the user can modify the contents).

**Valid values**

true(), false() or a valid expression.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page. The weblet will reserve a minimum width based on the data to be displayed.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

**autoSize**

Set to true to automatically resize the input field to accomodate dates in the current date format.

**Default value**

False

**Valid values**

true(), false() or a valid expression.

## **showAnim**

Sets the name of the animation used to show/hide the timepicker.

### **Default value**

show

### **Valid values**

show, slideDown or fadeIn

**duration**

Controls the speed at which the timepicker appears. Choose one of three predefined speeds.

**Default value**

normal

**Valid values**

slow, normal or fast

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

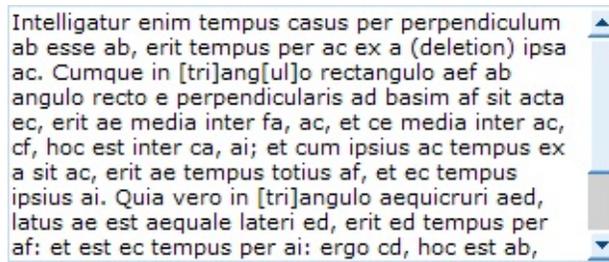
### **Valid values**

Any valid JavaScript statement(s).

### 8.3.10 Varchar (std\_varchar)

[QuickStart - Varchar](#)   [Properties - Varchar](#)

The varchar weblet provides a text input box control. It broadly corresponds to the `<input type="text">` HTML element. The weblet looks like this (when in 'memo' mode):



The varchar weblet is used to display and receive input for varying length character data. It is the default weblet for varying-length character fields.

## **QuickStart - Varchar**

To use this weblet to visualize a field that is already present on your web page, open the XSL for your webroutine in the LANSAs Editor and follow the following steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Varchar weblet.
2. Drag and drop the weblet over the existing field. Click on the weblet and then click on the Details tab.
3. If you dropped the weblet on an existing field, the name, value and other properties have already been set appropriately. Otherwise, set these properties now as required to associate the weblet with the required field in your webroutines web\_map.
4. If you wish the weblet to act as a multi-line input box, select 'memo' for the type property.

## Properties - Varchar

The Varchar weblet's properties are:

class	keyboard_shift	title
disabled	maxlength	type
display_length	name	value
display_mode	pos_absolute	width
height	read_only	word_wrap
hide_if	tab_index	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in a dark blue box, and its value is '#V1STR01'.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in a light blue box, and its value is 'key('field-value', 'V1STR01')'.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**maxlength**

Specifies the maximum number of characters the user can type in the weblet. When the weblet visualizes a field, this is set to the number appropriate for the field.

**Default value**

Blank (the weblet does not restrict the number of characters the user can type).

**Valid values**

A numeric value.

## **display\_length**

The approximate width of the weblet input box in characters – the browser sizes the input box according to the number of characters specified. If the width property is specified, it takes precedence and the display\_length property is ignored.

### **Default value**

Blank (the weblet assumes a default size).

### **Valid values**

A numeric value.

## **type**

Specifies the type of input control the weblet implements. This weblet is designed to implement an `<input type=text>` or an `<input type=password>` control or a `<textarea>` control. Other types are possible but are not supported for this weblet.

## **Default value**

'text'

## **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

- 'text'**        Creates a text entry control.
- 'memo'**        Creates a multi-line text entry control with word-wrapping.
- 'password'**    Creates a text entry control in which characters output or typed are not visible – instead an asterisk or other placeholder character is shown.

## **keyboard\_shift**

The keyboard shift for the input field.

### **Default value**

The keyboard shift of the field with this weblet visualization. Blank otherwise.

### **Valid values**

' ', 'W', 'J', 'E', 'O' and 'U'

The keyboard shift is currently only used to validate DBCS fields.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

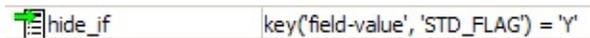
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value '#STD\_FLAG = 'Y'' is entered in a text field.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is visible in a text field. To its right, the value 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in another text field.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies a title for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **word\_wrap**

If 'memo' is specified for the type property, this property specifies how the weblet should handle word-wrapping when typing text.

### **Default value**

Blank.

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

- 'soft'** Text is displayed with word wrapping and submitted without carriage returns and line feeds.
- 'hard'** Text is displayed with word wrapping and submitted with soft returns and line feeds.
- 'off'** Word wrapping is disabled. The lines appear exactly as the user types them.

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

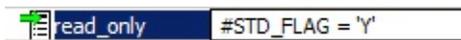
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width and height properties. However you can directly edit the property values if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default height).

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## 8.4 Layout Weblets

### 8.4.1 QuickStart - Standard Layouts

Layout weblets provide the basic HTML page structure (head, title, body, etc.) needed by all HTML pages. They also provide standard resources required by all webrouines (scripts, styles, hidden fields, etc.) and standard layout elements commonly used on web pages (headers, footers, content areas, sidebars, etc.)

Layout weblets are the base of WAM layouts. They provide a common interface across a WAM or even a complete web application. Typically layouts provide a consistent look and feel to the application by implementing menus, color schemes, message handling, images, logos, trademark information etc.

By default a WAM will use a single WAM layout across all generated webrouines. To share common layout features you can either use the same site layout (refer to [How do I Create my Own Site Layout?](#) for further information) for your whole application or base your WAM layouts on a common layout weblet. The generated WAM layout will share the common layout features from your site layout or common layout weblet and may then be customized for specific WAM requirements. Refer to [WAM Layouts and Layout Weblets](#) for further information.

The Layout Weblet group includes the standard layout weblets shipped with LANSA. WAM Layouts, which follow the naming convention <WAMname>\_layout, will, by default, be included in a different group, WAM Layouts.

Never modify these shipped layouts. If you wish to create a variation, create a copy and change the copy.

<b>Weblet name</b>	<b>Description</b>
<a href="#">Standard Theme Layouts (std_themelet1_[1-3]col)</a>	Basic jQuery themed layouts for your Webroutine pages. Each layout has 1, 2 or 3 content areas. std_themelet1_1col is the default layout.
<a href="#">Standard Blank Layout (std_blank_layout)</a>	A simple blank layout with no layout/css.
<a href="#">Standard Basic Layout</a>	A basic layout with header and footer, but

(std\_layout\_V2 and  
std\_layout[1-5]\_v2)

no menus.  
and  
basic layouts, each providing a different  
appearance for your Webroutine pages.

## 8.4.1 QuickStart - Standard Layouts

Typically you will develop or customize a layout weblet to suit your own site standards or application.

You may choose to use the Site Layout Manager Wizard to help you create your site layout weblet (Refer to [How do I Create my Own Site Layout?](#) for further information).

To use your site layout weblet as the base for all the WAM layouts in your application, specify your site layout weblet as the Layout Weblet when creating each of your WAMs. This is the recommended approach.

Alternatively, to apply your layout to a WAM layout you may follow these steps:

1. Open the WAM layout to which you want to apply the layout weblet.
2. Select Weblet templates in the repository view and select Layout weblets from the Weblet groups dropdown list.
3. Find the layout weblet you want to apply from the repository view.
4. Drag and drop the layout weblet on to the Design view of your WAM layout.
5. The layout weblet will be added to the existing layout.
6. Set focus on the old layout and press the delete key. You will be left only with the new layout.
7. Save your WAM layout
8. If this is a customized layout for your site this is probably all you need to do.

Don't drop the layout weblet directly on to the design of the WebRoutine. If you do, the layout changes you make to the WebRoutine design will only apply to that WebRoutine, not to the entire WAM.

Note that the logo images used in the layout are defined in the std\_style.min.css Cascading Style Sheet in the std\_headerl and std\_headerr classes.

## 8.4.2 Standard Theme Layouts (std\_themelet1\_[1-3]col)

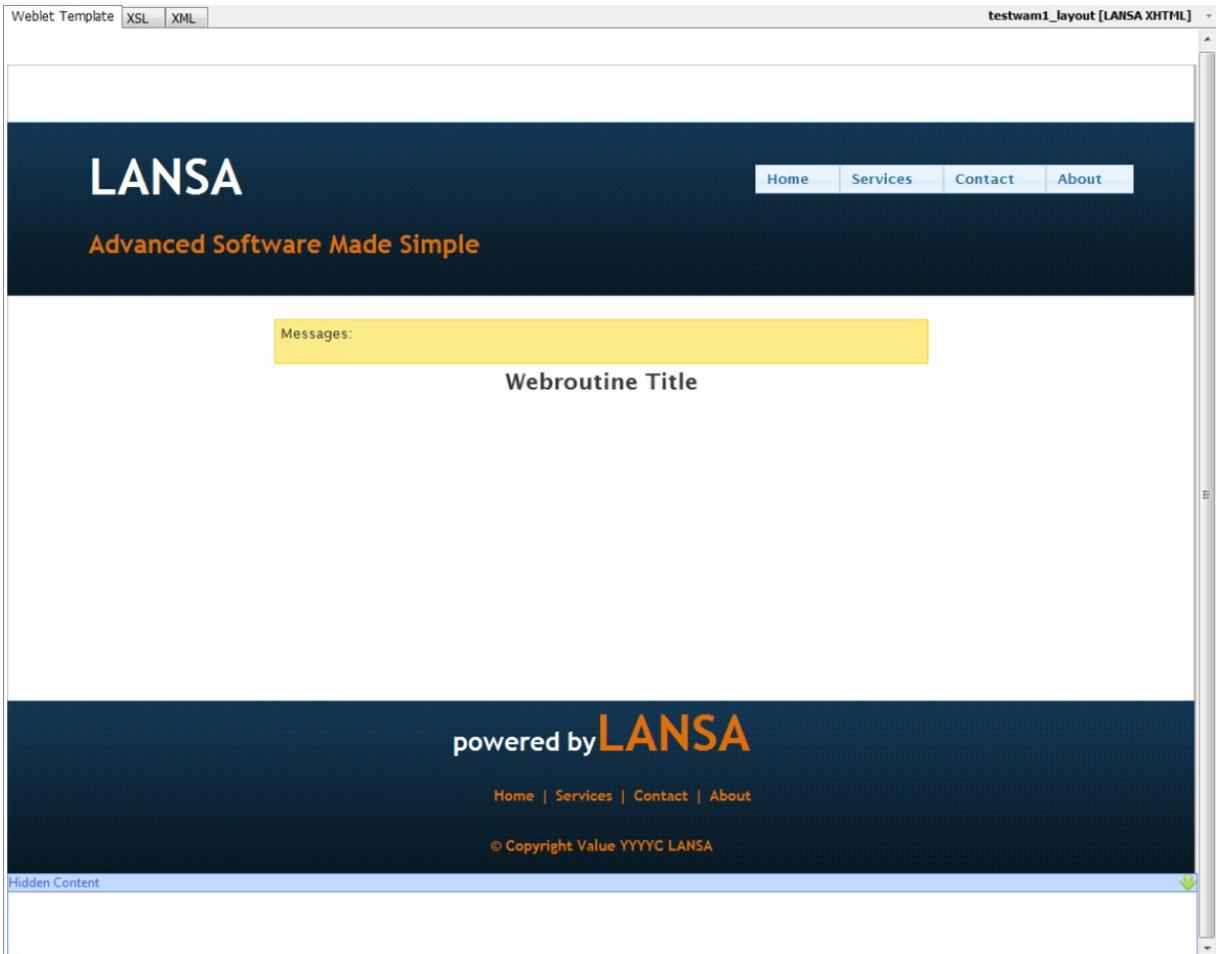
**std\_themelet1\_[1-3]col** are jQuery themed layouts with header, footer, content and hidden areas defined.

The three layouts are essentially the same but provide 1, 2 or 3 content areas and can be either fixed or fluid width. There are currently 2 styles and 8 jQuery themes provided that can be applied to these layouts.

Use the arrow to the far right of the Hidden Content area to collapse and later expand the Hidden Content area.

**std\_themelet1\_1col** is the default layout used for the auto-generated WAM layout when LANSA generates the WAM XSL. The default is applied if a specific site layout weblet has not been specified when the WAM was created. The std\_themelet1\_1col layout weblet must never be modified.

A WAM layout that has been created using the std\_themelet1\_1col layout weblet looks like this when opened in the Editor (using Redmond theme and Style #1 Themelet):

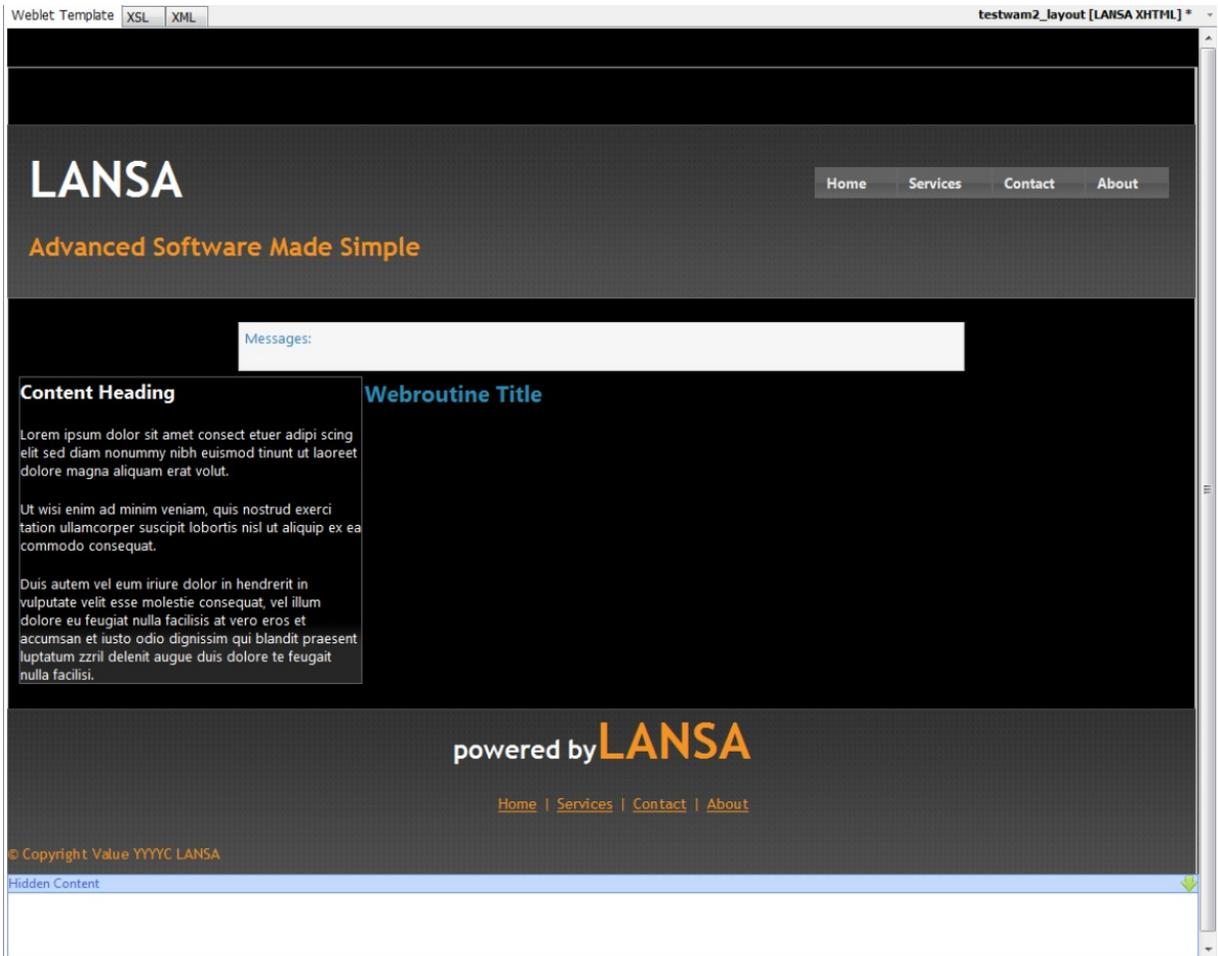


As you can see from this image the layout generally controls the header and footer, the messages area, content area and the color theme and style for all webroutine pages using the layout. JavaScript can also be applied at the layout level and will then apply to all webroutines using the layout.

The Web\_Maps and weblets associated with a specific Webroutine will appear in the content area of the layout (below the Webroutine Title).

std\_themelet1\_2col layout weblet is essentially the same as std\_themelet1\_1col layout weblet but with two content areas. The std\_themelet1\_2col layout weblet must never be modified.

A WAM layout that has been created using the std\_themelet1\_2col layout weblet looks like this when opened in the Editor (using Darkness theme and Style #2 Themelet):



std\_themelet1\_3col layout weblet is essentially the same as std\_themelet1\_1col and std\_themelet1\_2col layout weblets but with three content areas. The std\_themelet1\_3col layout weblet must never be modified.

A WAM layout that has been created using the std\_themelet1\_3col layout weblet looks like this when opened in the Editor (using South Street theme and Style #1 Themelet):

# LANSA

Advanced Software Made Simple

[Home](#) [Services](#) [Contact](#) [About](#)

Messages:

Menu Link

Menu Link

Menu Link

Menu Link

Menu Link

## Webroutine Title

### Content Heading

Lorem ipsum dolor sit amet consectetur adipiscing elit sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volut.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

powered by **LANSA**

[Home](#) | [Services](#) | [Contact](#) | [About](#)

## Properties - Standard Theme Layouts (std\_themelet1\_[1-3]col)

The Standard Themelet Layouts (std\_themelet1\_[1-3]col) weblet properties are:

Backcompat\_theme    jQueryNoConflict    sidebar2\_width  
content\_side        output\_charset        title\_text  
content\_width        show\_title            width  
css\_files            sidebar\_width         width\_type  
has\_form             sidebar1\_width        window\_title  
javascript\_files

## Backcompat\_theme

A Boolean property that determines if the theme passed to `std_style` is "" or 'none'. If evaluated to be True, a backwards compatible theme is applied so that the layout and weblets will be consistent with existing applications prior to V12 SP1. If False, no theme will be applied, as External Resources are now the method used to apply themes.

### Default value

False.

### Valid values

True or False.

### Example

```
backcompat_theme False
```

## **content\_side**

The location of the main content area in the WAM layout relative to other content areas. This is only applicable to either the 2 or 3 column layouts.

### **Default value**

Center.

### **Valid values**

Left, Center or Right.

### **Example**

 content\_side *center*

## **content\_width**

A value specified in CSS units for the main content width. This may be a relative, absolute or percentage value. This is only applicable to either the 2 or 3 column layouts.

### **Default value**

70% for 2 column layout and 50% for 3 column layout.

### **Valid values**

A width, in a valid unit of measurement, in single quotes..

### **Example**

`content_width` `50%`

## **css\_files**

A comma separated list of Cascading Stylesheet (CSS) files to be applied to the layout. These files will be applied the file defined in theme\_css\_filename This property is designed to allow for special circumstances when one stylesheet isn't enough.

Refer to [Cascading Style Sheets \(CSS\) and the Style weblet](#) for more information on the use of CSS in your layouts.

### **Default value**

"

### **Valid values**

A comma separated list of path and CSS filename values, relative to the style virtual directory, enclosed in single quotes.

## has\_form

A Boolean property. An expression which, if evaluated to be False, will remove the default LANSA form element.

### Default value

true() (that is, the LANSA form element is included in the page)

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will remove the default LANSA form if field #HAS\_FORM is equal to 'N'. The expression should be entered, and is shown when the property has focus, as follows:



has\_form #has\_form = 'N'

When the property loses focus, the expression is shown as follows:



has\_form key('field-value', 'HAS\_FORM') = 'N'

## **output\_charset**

The character set encoding identifier of the output document, such as iso-8859-1 or windows-1251. Some character sets must be converted before they can be used in, for example, email or PDF documents. If the default charset is one of them, this attribute will contain the name of the character set output will be converted to. You should normally leave this blank so a default value is used.

### **Default value**

/lxml:data/lxml:server-instructions/lxml:client-charset

### **Valid values**

Single quote character set value.

## **show\_title**

A Boolean property. An expression which, if evaluated to be True, will show the title.

### **Default value**

true() (that is, the grid will always be shown)

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

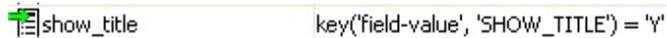
### **Example**

This example will show the title if field #SHOW\_TITLE is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_title'. To its right is a blue rectangular field containing the expression '#SHOW\_TITLE = 'Y''. A vertical line separates the field from the rest of the editor.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_title'. To its right is a light gray rectangular field containing the expression 'key('field-value', 'SHOW\_TITLE') = 'Y''. A vertical line separates the field from the rest of the editor.

**title\_text**

The title text to be displayed on the webroutine page.

**Default value**

/lxml:data/lxml:context/lxml:webroutine-title (effectively Blank).

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## **javascript\_files**

A comma separated list of JavaScript files to be loaded into the layout. This property allows you to include your own custom JavaScript or code associated with third party controls.

Refer to [JavaScript and the Script weblet](#) for more information on the use of JavaScript in your layouts.

### **Default value**

"

### **Valid values**

A comma separated list of path and JavaScript filename values, relative to the script virtual directory, enclosed in single quotes.

## **width**

A value specified in CSS units for the page width. This may be a relative, absolute or percentage value.

### **Default value**

1000px

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

### **Example**

 width *1000px*

## **width\_type**

The CSS layout type.

A fixed width layout is one where the width is set to a specific value that is independent of the size of the client window. Layout content will not be adjusted as the window is resized.

A fluid width layout is one where the layout content adjusts itself to fit the width of the client window. For example, a 100% width fluid layout will take up the entire width without producing a scroll bar. It will only produce a scroll bar when resized to a small window.

### **Default value**

Fixed for the 1 column layout and Fluid for the 2 and 3 column layouts.

### **Valid values**

Fixed or Fluid.

### **Example**

 width\_type *Fixed*

## **sidebar\_width**

A value specified in CSS units for the width of the sidebar content area of the 2 column layout. This may be a relative, absolute or percentage value.

### **Default value**

29%

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

### **Example**

 sidebar\_width 29%

## **sidebar1\_width**

A value specified in CSS units for the width of the sidebar1 content area of the 3 column layout. This may be a relative, absolute or percentage value.

### **Default value**

19%

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

### **Example**

 sidebar1\_width 19%

## **sidebar2\_width**

A value specified in CSS units for the width of the sidebar2 content area of the 3 column layout. This may be a relative, absolute or percentage value.

### **Default value**

30%

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

### **Example**

 sidebar2\_width 30%

## jQueryNoConflict

A Boolean property that only applies if you use jQuery. If evaluated to be True, jQuery.noConflict() is called to relinquish the \$ name. You need to include the other library that uses the \$ name before you include jQuery Core.

### Default value

False.

### Valid values

True or False.

### Example

```
jQueryNoConflict False
```

## **window\_title**

The text for the title that will appear in the client window for the web page.

### **Default value**

The description for the WAM.

### **Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

### **Example**

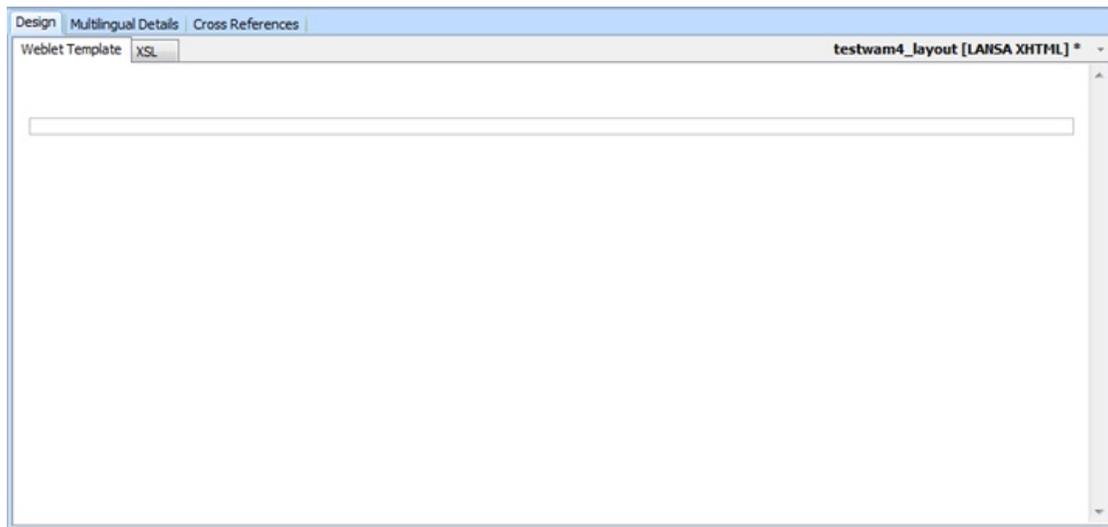
 window\_title *\$lweb\_context/bxml:webapplication-title*

### 8.4.3 Standard Blank Layout (std\_blank\_layout)

The std\_blank\_layout layout weblet is a plain layout with no header or footer areas defined and no color scheme. That is, it is used for displaying a Webroutine with no layout/css.

The std\_blank\_layout layout weblet must never be modified.

A WAM layout that has been created using the std\_blank\_layout layout weblet looks like this when opened in the Editor:



With the introduction of this new blank layout, the method for displaying a WebRoutine with no layout/css has been simplified. Instead of selecting the layout weblet part of the design view, and changing the property no\_layout to true you simply:

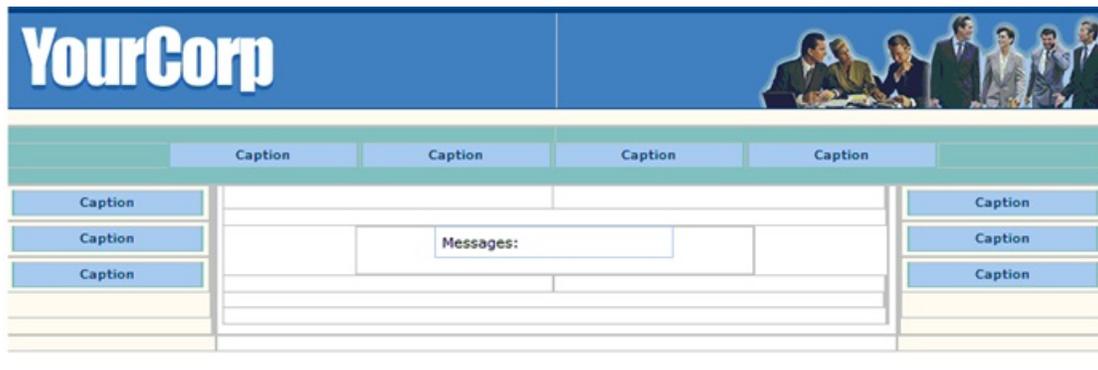
drag this std\_blank\_layout weblet onto the webroutine. It will replace whatever layout it had previously.

## 8.4.4 Standard Basic Layout (std\_layout\_V2 and std\_layout[1-5]\_v2)

Std\_layout\_v2 is a basic layout with a header and footer areas defined.

Standard layouts 1-5 are essentially the same but have different settings applied to determine the menus to be displayed (std\_layout1\_v2, std\_layout2\_v2 ... std\_layout5\_v2).

When opened in the editor the std\_layout1 weblet looks like this:



As you can see from this image the layout generally controls the menus, banners and images, the message box and the color scheme for pages using the layout. JavaScripts can also be applied at the layout level and will then apply to all webrouines using the layout.

The WEB\_MAPs and weblets associated with a specific webroutine will appear in the body of the layout (below the message box).

## Properties - Standard Layouts(std\_layout\_V2 and std\_layout[1-5]\_v2)

The Standard Layouts weblet's properties are:

body_class	onload_script	show_title
css_files	onunload_script	show_top_menu
form_class	output_charset	theme_css_filename
has_form	show_left_menu	title_text
javascript_files	show_messages	trap_script_errors
no_layout	show_right_menu	

## show\_left\_menu

*Not available on std\_layout.*

A Boolean property. An expression which, if evaluated to be True, will show the left menu.

### Default value

Varies with layout.

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

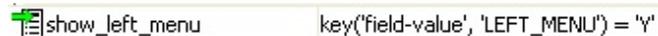
### Example

This example will show the left menu if field #LEFT\_MENU is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_left\_menu'. To its right, a blue rectangular box contains the expression '#LEFT\_MENU = 'Y''. The entire row is underlined.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_left\_menu'. To its right, a light gray rectangular box contains the expression 'key('field-value', 'LEFT\_MENU') = 'Y''. The entire row is underlined.

## **show\_top\_menu**

*Not available on std\_layout.*

A Boolean property. An expression which, if evaluated to be True, will show the top menu.

### **Default value**

Varies with layout.

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

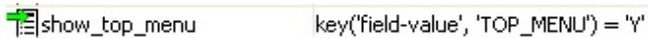
### **Example**

This example will show the top menu if field #TOP\_MENU is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a label 'show\_top\_menu' with a small icon to its left. To the right of the label is a text input field containing the expression '#TOP\_MENU = 'Y''. The input field has a blue border and a small cursor at the end of the text.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a label 'show\_top\_menu' with a small icon to its left. To the right of the label is a text input field containing the expression 'key('field-value', 'TOP\_MENU') = 'Y''. The input field has a light gray border and a small cursor at the end of the text.

## **show\_right\_menu**

*Not available on std\_layout.*

A Boolean property. An expression which, if evaluated to be True, will show the right menu.

### **Default value**

Varies with layout.

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

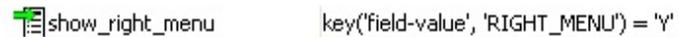
### **Example**

This example will show the right menu if field #RIGHT\_MENU is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_right\_menu'. To its right is a text input field containing the expression '#RIGHT\_MENU = 'Y''.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_right\_menu'. To its right is a text input field containing the expression 'key('field-value', 'RIGHT\_MENU') = 'Y''.

## **body\_class**

The Cascading Style Sheet class to be applied to the body element of the page.

### **Default value**

Blank.

### **Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

### **Example**

 body\_class      'redbg'

**form\_class**

The Cascading Style Sheet class to be applied to the form element of the page.

**Default value**

Blank.

**Valid values**

Any valid class name from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## has\_form

*Not available on std\_layout.*

A Boolean property. An expression which, if evaluated to be False, will remove the default LANSA form element.

### Default value

true() (that is, the LANSA form element is included in the page)

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

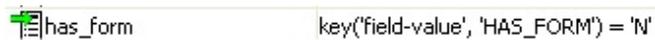
### Example

This example will remove the default LANSA form if field #HAS\_FORM is equal to 'N'. The expression should be entered, and is shown when the property has focus, as follows:

A screenshot of a software interface showing a property editor for 'has\_form'. The property name 'has\_form' is on the left, and the expression '#has\_form = 'N'' is entered in a text field on the right. The text field is highlighted with a blue background, indicating it has focus.

has\_form #has\_form = 'N'

When the property loses focus, the expression is shown as follows:

A screenshot of the same software interface. The property name 'has\_form' is on the left, and the expression 'key('field-value', 'HAS\_FORM') = 'N'' is shown in the text field. The text field is no longer highlighted, indicating it has lost focus.

has\_form key('field-value', 'HAS\_FORM') = 'N'

## **no\_layout**

*Not available on std\_layout.*

A Boolean property. An expression which, if evaluated to be True, will remove the layout outline.

### **Default value**

false() (that is, the layout outline is include in the page)

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### **Example**

This example will remove the layout outline if field #NO\_LAYOUT is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a label 'no\_layout' with a small icon to its left. To the right of the label is a text input field containing the expression '#no\_layout = 'Y''. The input field has a blue highlight on its left side.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a label 'no\_layout' with a small icon to its left. To the right of the label is a text input field containing the expression 'key('field-value', 'NO\_LAYOUT') = 'Y''. The input field has a light gray highlight on its left side.

## **show\_title**

A Boolean property. An expression which, if evaluated to be True, will show the title.

### **Default value**

true() (that is, the grid will always be shown)

### **Valid values**

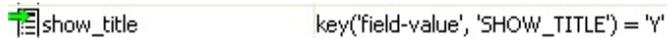
true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### **Example**

This example will show the title if field #SHOW\_TITLE is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:

A screenshot of a property editor. On the left, there is a small icon and the text 'show\_title'. A blue horizontal bar highlights this text. To the right of this bar, the expression '#SHOW\_TITLE = 'Y'' is displayed in a light gray box.

When the property loses focus, the expression is shown as follows:

A screenshot of a property editor. On the left, there is a small icon and the text 'show\_title'. To the right of this text, the expression 'key('field-value', 'SHOW\_TITLE') = 'Y'' is displayed in a light gray box.

**title\_text**

The title text to be displayed on the webroutine page.

**Default value**

/lxml:data/lxml:context/lxml:webroutine-title (effectively Blank).

**Valid values**

Single-quoted text or the name of a multilingual text variable, system variable or field (the ellipses button in the property sheet can be clicked to choose from a list).

## **show\_messages**

A Boolean property. An expression which, if evaluated to be True, will show the message box on the webroutine page.

### **Default value**

true() (that is, the message box will always be shown)

### **Valid values**

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

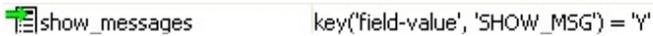
### **Example**

This example will show the message box if field #SHOW\_MESSAGES is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_messages'. To its right is a blue rectangular field containing the expression '#show\_msg = "Y"'. The entire field is highlighted with a blue background.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor. On the left, there is a small icon and the text 'show\_messages'. To its right is a white rectangular field containing the expression 'key("field-value", "SHOW\_MSG") = "Y"'. The field has a thin border.

## **onload\_script**

*Not available on std\_layout.*

JavaScript to execute after the page is loaded into the browser (that is, body tag onload event).

### **Default value**

'SetFocus()' (that is, When the page is loaded execute the function SetFocus() which is included in the standard script.)

### **Valid values**

Single-quoted valid JavaScript function, or JavaScript code followed by a semicolon (;).

## **onunload\_script**

*Not available on std\_layout.*

JavaScript to execute after the page is unloaded from the browser (that is, body tag onunload event).

### **Default value**

"

### **Valid values**

Single quoted valid JavaScript function, or JavaScript code followed by a semicolon (;).

## **javascript\_files**

A comma separated list of JavaScript files to be loaded into the layout. This property allows you to include your own custom JavaScript or code associated with third party controls.

Refer to [JavaScript and the Script weblet](#) for more information on the use of JavaScript in your layouts.

### **Default value**

"

### **Valid values**

A comma separated list of path and JavaScript filename values, relative to the script virtual directory, enclosed in single quotes.

## **theme\_css\_filename**

Cascading Stylesheet (CSS) filename, relative to the style virtual directory, to be applied to the layout and webllets dependant on the layout. This will determine the style properties for HTML elements and the CSS class selectors available for selection on subordinate webllets' class properties.

Refer to [Cascading Style Sheets \(CSS\) and the Style webllet](#) for more information on the use of CSS in your layouts.

### **Default value**

"

### **Valid values**

Any path and CSS filename, relative to the style virtual directory, enclosed in single quotes. A file can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **css\_files**

A comma separated list of Cascading Stylesheet (CSS) files to be applied to the layout. These files will be applied the file defined in theme\_css\_filename This property is designed to allow for special circumstances when one stylesheet isn't enough.

Refer to [Cascading Style Sheets \(CSS\) and the Style weblet](#) for more information on the use of CSS in your layouts.

### **Default value**

"

### **Valid values**

A comma separated list of path and CSS filename values, relative to the style virtual directory, enclosed in single quotes.

## **output\_charset**

The character set encoding identifier of the output document, such as iso-8859-1 or windows-1251. Some character sets must be converted before they can be used in, for example, email or PDF documents. If the default charset is one of them, this attribute will contain the name of the character set output will be converted to. You should normally leave this blank so a default value is used.

### **Default value**

/lxml:data/lxml:server-instructions/lxml:client-charset

### **Valid values**

Single quote character set value.

## trap\_script\_errors

A Boolean property. An expression which if evaluated to be True will suppress any JavaScript errors.

### Default value

true() (that is, JavaScript errors will be suppressed)

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

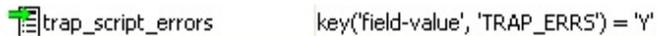
### Example

This example will suppress any JavaScript errors if the field #TRAP\_ERRS is equal to 'Y'. The expression should be entered, and is shown when the property has focus, as follows:

A screenshot of a form field. The field name is 'trap\_script\_errors' and the value is '#TRAP\_ERRS = 'Y''. The field is highlighted with a blue background, indicating it has focus.

trap\_script\_errors #TRAP\_ERRS = 'Y'

When the property loses focus, the expression is shown as follows:

A screenshot of a form field. The field name is 'trap\_script\_errors' and the value is 'key('field-value', 'TRAP\_ERRS') = 'Y''. The field is highlighted with a yellow background, indicating it has lost focus.

trap\_script\_errors key('field-value', 'TRAP\_ERRS') = 'Y'

## 8.4.5 Utility Weblets

<b>Weblet name</b>	<b>Description</b>
Standard hidden fields (std_hidden)	For LANSA Product Centre's internal use only. Refer to <a href="#">Hidden</a> for information.
Standard locale (std_locale)	For LANSA Product Centre's internal use only. Refer to <a href="#">Local</a> for information.
Standard script (std_script)	For LANSA Product Centre's internal use only. Refer to <a href="#">JavaScript and the Script Weblet</a> for information.
Standard variables (std_variables)	For LANSA Product Centre's internal use only. Refer to <a href="#">Variables</a> for information.
std_keys	A set of xsl:key elements used by field references in XSL. Refer to <a href="#">Keys</a> for information.
std_select_list	A weblet used by std_dropdown and std_listbox. Should not be used directly.
std_types	For LANSA Product Centre's internal use only. Refer to <a href="#">Types</a> for information.
std_util	For LANSA Product Centre's internal use only.
CSS Styles (std_style_v2)	Default style that is used for all generated WEBROUTINE pages.

## **8.4.6 Inline Templates**

The Inline Templates group is provided to allow quick access to all webllets which are "inline-aware" for the current Technology Service. "Inline-aware" webllets include a weblet template with the suffix `.inline`.

All Webllets included in the Inline Templates group are also available in the [8.1 Standard Webllets](#) or [8.3 Standard Field Visualizations](#) groups.

## 8.4.7 jQuery UI

The jQuery UI group is provided to allow quick access to all weblets which are used and require jQuery-ui to work. All Weblets included in the jQuery UI group are also available in the [8.1 Standard Weblets](#) and [8.3 Standard Field Visualizations](#).

## 8.4.8 WAM Layouts

A WAM layout, with the name <WAMname>\_layout will be created when XSL is generated for a WAM unless it already exists. Use the LAYOUTWEBLET selector on the BEGIN\_COM command of the WAM. to nominate the layout weblet to be used by the WAM layout. The generated WAM layout is included in the Weblet Template group WAM Layouts.

Refer to [WAM Layouts and Layout Weblets](#) for further information.

## 9. Weblets for jqMobile Technology Service

jQuery Mobile is a unified, HTML5-based user interface system for all popular mobile device platforms, built on a jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement, and has a flexible, easily themeable design.

The jqMobile Technology service is a wrapper around the JavaScript library that allows it to work in the LANSA Editor with drag-and-droppable weblets that are configured by editing properties. This section documents the Weblet Templates that are shipped with the jqMobile Technology Service.

The Weblet Templates repository view displays templates by group. A weblet template may belong to more than one group (right-click a weblet to configure). The standard groups are:

[Standard Weblets](#)

[Layout Weblets](#)

### **Also see**

[QMobile and the WAM Editor](#)

[Field Validation](#)

[Default Weblet](#)

[Utility Weblets](#)

## 9.1 jQuery Mobile and the WAM Editor

The jQuery Mobile framework relies heavily on JavaScript to do all its work in the browser at runtime. It takes minimal HTML as its starting point and uses JavaScript to modify the HTML according to the needs and capabilities of the browser it is running on.

This runtime enhancement presents a problem for the concept of graphical editing that you may be used to from other Technology Services like XHTML. The WAM Editor cannot execute the runtime JavaScript so it is only able to display and edit the basic HTML that provides the starting point for jQuery Mobile.

Many of the weblets are container weblets. That is, they act as wrappers for custom content and server to modify how that custom content appears or behaves. The Header and Footer weblets are examples of this. They define certain positioning and visual characteristics for a header or footer but allow you to define the actual content that goes into the header or footer. This content may be some simple text, other weblets, or your own custom HTML. Container weblets cannot be selected in the designer by clicking on them because, when you click on them, you are actually selecting the content. To select these weblets and edit their properties you will need to use the outline tab.

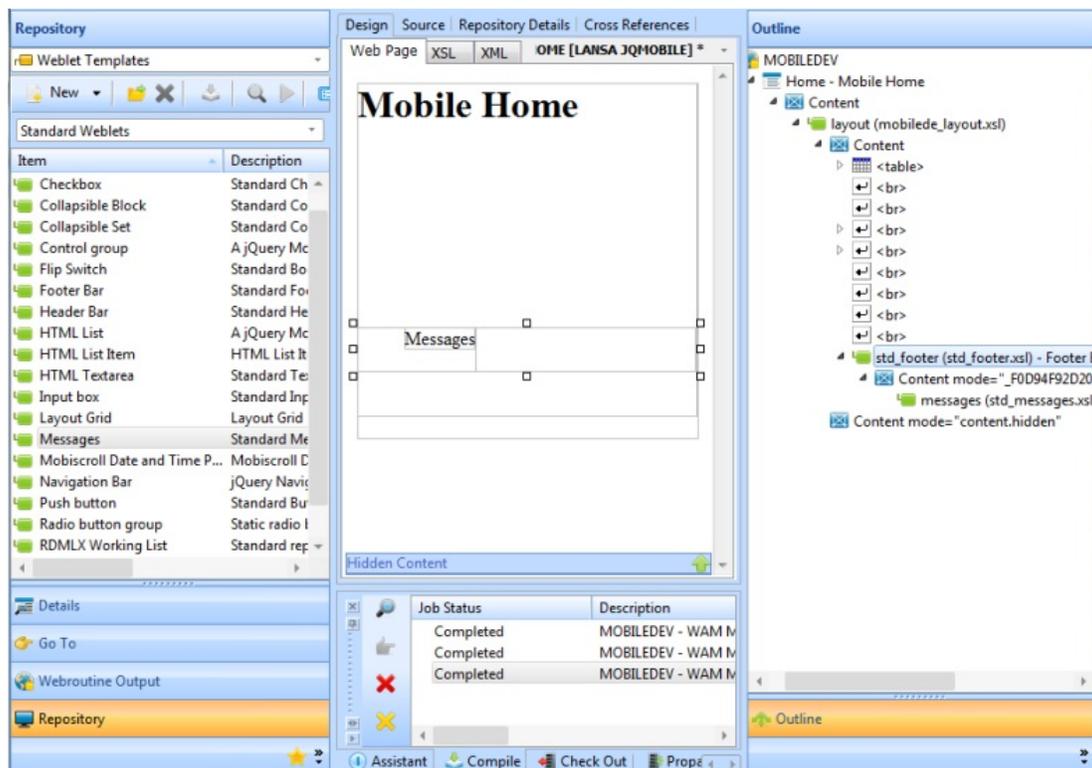


Figure 9.1.a Using the outline to select the Footer weblet

## 9.2 Field Validation

The HTML 5 specification includes a number of features for form validation. Form elements have a number of new attributes like required, min, max, pattern and new data types that allow the browser to either present a UI that prevents invalid data entry or validate the entered data and prevent the form being submitted if it is incorrect. The LANSAs framework includes some enhancements that add support for RDMLX data types to the validation, add some extra functionality and address some inconsistencies in browser implementation.

### Also see

[9.2.1 RDMLX Data types](#)

[9.2.2 Displaying Validation Errors](#)

[9.2.3 Controlling When Validation Occurs](#)

## 9.2.1 RDMLX Data types

Validation of RDMLX data types can be added to any `<input>`, `<select>` or `<textarea>` element by adding a `data-lstddatatype` attribute to it. The attribute value should be a | delimited string starting with the data type followed by extra parameters as required by the data type:

`integer|<max length>`

`float|<max length>`

`packed|<total digits>|<fraction digits>|<decimal separator>`

`signed|<total digits>|<fraction digits>|<decimal separator>`

`dec|<total digits>|<fraction digits>|<decimal separator>`

`alpha|<keyboard shift>|<max length>`

`char|<keyboard shift>|<max length>`

`varchar|<keyboard shift>|<max length>`

`nchar|<keyboard shift>|<max length>`

`nvarchar|<keyboard shift>|<max length>`

For example, a field defined with the following RDMLX:

```
Define Field(#TST_PKD) Type(*PACKED) Length(6) Decimals(2)
```

might use an `<input>` tag like this:

```
<input id="MyWR_TST_PKD" name="TST_PKD" maxlength="6" size="11"  
      data-lstddatatype="packed|6|2|." type="number" />
```

When you generate a new webroutine or drop a field onto a design, the WAM Editor will automatically set this attribute for you. Standard weblets that create form elements will set this attribute based on the value of their `rdmlxDataType` property.

## 9.2.2 Displaying Validation Errors

Most modern desktop browsers will display validation errors by displaying a tooltip over the field containing an error message. Usually, only one error is displayed at a time (requiring another submit to check for further errors) and there is no option to display errors at other times, such as during typing or when the user tabs out of the field. At the time of writing, mobile device browsers did not do even this. While they will not allow an invalid form to be submitted, they do not display any errors to explain the problem.

The LANSAs framework adds the option of displaying validation errors within the page, usually next to the field. To enable this behavior, you must add the class "lstdErrorShowInDiv" to the page <div> in your layout. The standard shipped layouts do this using the validationErrorDisplay property. You must also provide the <div> tags that will display the error. There must be one error <div> for each field. The <div> must have an ID of "{Field ID}\_id" and a class of "lstdFieldError". For example:

```
<div id=" MyWR_TST_PKD _error" class="lstdFieldError"></div>
```

The standard field weblets will do this for you via the addErrorDiv property. When the "show in div" behavior is turned off, all error divs are automatically hidden. When the "show in div" behavior is turned on, all error divs are set to invisible, meaning that they cannot be seen by the user but space is still reserved for them in the document. This prevents the confusion of the page re-flowing if they are later made visible. When a validation error occurs in a field the framework will place the error message inside the corresponding error div and make it visible.

### 9.2.3 Controlling When Validation Occurs

The default browser behavior is to perform validation when the user submits the form. The LANSAs framework offers two options to extend this: after the user has entered data in a field and moved the focus away, and immediately while the user is typing. To enable these options, add one of the following classes to the page `<div>`: `lstdErrorImmediately` or `lstdErrorAfterFocus`. The standard shipped layouts do this using the `validationTime` property.

While most validation can be performed at these other times, some browsers may have exceptions for specific scenarios. For example, Safari will not invalidate a required field after focus unless the user has made some attempt to enter data in it. This allows a user to tab through fields while filling in a form without being hit with errors along the way.

### **9.3 Default Weblet**

jQuery Mobile has features to allow for responsive designs. They rely on input fields having special wrapper fields and jQuery Mobile data attributes. This is easier to enforce with a weblet. For this reason, the JQMOBILE Technology Service defines `std_input` as the default weblet visualization for any repository field that doesn't have one.

## 9.4 Standard Weblets

Following is a description of the standard weblets in alpha sequence, their properties and how to use them in your own webrouines. You will not use every property available for a weblet.

<a href="#">Anchor</a> ( <a href="#">std_anchor</a> and <a href="#">std_anchor_s1</a> )	Provides a hyperlink (or anchor) control and broadly corresponds to the HTML element that designates the destination of a hypertext link.
<a href="#">Autocomplete</a> ( <a href="#">std_autocomplete</a> )	While you type, the Autocomplete weblet provides suggestions provided by a WebRoutine using Ajax.
<a href="#">Boolean</a> ( <a href="#">std_boolean</a> )	This weblet generates an HTML element that jQuery Mobile converts into a sliding switch.
<a href="#">Button</a> ( <a href="#">std_button_s1</a> and <a href="#">std_button_v2</a> )	Represents creates a clickable button and can contain custom HTML like formatted text and images (unlike a button created with the Input box weblet).
<a href="#">Checkbox</a> ( <a href="#">std_checkbox</a> )	A standard checkbox control which jQuery Mobile will modify to fit the current theme.
<a href="#">Collapsible Block</a> ( <a href="#">std_collapsible</a> )	A section of content that the user can show or hide by clicking on its header.
<a href="#">Collapsible Set</a> ( <a href="#">std_collapsibleset</a> )	A wrapper that can be placed around multiple Collapsible Block weblets to create an accordian. You create the Collapsible Block weblets and place them into the content area of the Collapsible Set weblet.
<a href="#">Control Group</a> ( <a href="#">std_controlgroup</a> )	A wrapper that can be placed around multiple buttons, checkboxes or radio buttons to group them together.
<a href="#">File Upload</a> ( <a href="#">std_fileupload</a> )	This weblet allows you to select files to upload to a temporary directory on the application server. The receiving webrouine can then manipulate the uploaded files.
<a href="#">Footer</a> ( <a href="#">std_footer</a> )	A jQuery Mobile toolbar at the bottom of the page. Can be "fixed" to remain in position when the page is scrolled or configured to show or hide itself when the user taps on

the screen.

[Header  
\(std\\_header\)](#)

A jQuery Mobile toolbar at the top of the page. Can be "fixed" at the top of the screen to remain in position when the page is scrolled or show or hide itself when the user taps on the screen.

[HTML List  
\(std\\_html\\_list\)](#)

A wide range of lists and formatting types that are used for data display, navigation, result lists, and data entry.

[HTML List Item  
\(std\\_html\\_li\)](#)

Adds items to a HTML List weblet.

[HTML Textarea  
\(std\\_textarea\)](#)

An element that can be used for the display and input of long text values spanning multiple lines.

[Image \(std\\_image\)](#)

Displays an image.

[Input Box  
\(std\\_input\)](#)

This weblet creates an element is the basic form of data entry in HTML pages. Used to create every type of input control (including buttons) that HTML supports except multi-line text fields.

[Layout Grid  
\(std\\_gridlayout\)](#)

Used if you need to place small elements side-by-side (like buttons or navigation tabs, for example).

[Loader  
\(std\\_loader\)](#)

A small loading overlay displayed when jQuery Mobile loads in content via AJAX, or for use in custom notifications.

[Messages  
\(std\\_messages\)](#)

Displays messages created with the RDMLX Message command.

[Mobiscroll Date and Time Picker  
\(std\\_mobiscroll\)](#)

A customizable date and time picker optimised for mobile devices.

[Navigation Bar  
\(std\\_navbar\)](#)

A very basic widget that can provide up to 5 buttons with optional icons.

[Progress bar  
\(std\\_progressbar\)](#)

Display status of a determinate process.

[Radio Button](#)

Creates a set of radio buttons linked to an RDMLX field.

Group  
(std\_radbuttons)

RDMLX Working List (std\_repeater) Provides a flexible mechanism to process working lists.

Select Menu (std\_dropdown) Based on a native `select` element, which is hidden from view and replaced with a custom-styled select button that matches the look and feel of the jQuery Mobile framework.

### 9.4.1 Anchor (std\_anchor and std\_anchor\_s1)

The anchor weblet provides a hyperlink (or anchor) control. It broadly corresponds to the <a> (anchor) HTML element that designates the destination of a hypertext link. Weblet std\_anchor has the same properties as std\_anchor\_s1, but with the difference that you can add content to the anchor element.

- In its simplest form the anchor weblet can display a text value that, when tapped will navigate the browser to a new page.
- The anchor weblet can be configured to display as a button, or it can contain any arbitrary HTML (including other weblets).
- The destination can be a url (such as <http://www.yourcompany.com/>) or you can specify a WAM and webroutine to be executed and optionally identify fields whose values should be passed to the webroutine.
- The anchor always generates a GET HTTP request and does not submit any form fields unless explicitly specified in the onclickExtraFields property. So the anchor should only be used for linking to external sites, static pages or webroutines that perform queries. When performing transactions that create or update data you should use a submit button.
- HTML 5 form validation is not performed on fields sent with an anchor click. If you need validation, use a submit button.

## Properties - Anchor (std\_anchor and std\_anchor\_s1)

Use std\_anchor only if you need to add content to the anchor element.  
For all other cases, use std\_anchor\_s1.

This weblet's properties are:

absoluteImagePath (std_anchor_s1 only)	icon iconPosition iconShadow	relativeImagePath (std_anchor_s1 only) shadow
class	id	style
corners	inline	swatch
customIcon (std_anchor_s1 only)	internal_id	tabindex
countIndicator (std_anchor_s1 only)]	mini	transition
countValue (std_anchor_s1 only)	onClickExtraFields	transitionDirection
displayAs	onClickWamName	url
hideIf	onClickWrName	useAjax
	relationship	value

### **absoluteImagePath (std\_anchor\_s1 only)**

The path and file name of the image to be displayed. If specified, the relative\_image\_path property should be left blank.

#### **Default value**

Blank – the default is to use the image specified in the relative\_image\_path property.

#### **Valid values**

The path and name of an image enclosed in single quotes.

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**value**

The text to display in the page for the anchor. The anchor weblet can contain any custom HTML content. When the weblet is first placed on a design, the WAM Editor automatically creates custom content that displays this value string. If you override the custom content, this property will no longer have any effect.

**Default value**

Blank.

**Valid values**

Any plain text string.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

## **corners**

Specifies if the button should have rounded corners.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and will be ignored otherwise.

## **Default value**

True

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**countIndicator (std\_anchor\_s1 only)]**

Shows a count bubble to the right of the list item if set to true.

**Default value**

False – Don't show count bubble.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**countValue (std\_anchor\_s1 only)**

The value to show in the count bubble.

**Default value**

None

**Valid values**

A numeric value.

**customIcon (std\_anchor\_s1 only)**

Shows the image (relative or absolute image path) as an icon if set to true.

**Default value**

False – Show image as a thumbnail.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **displayAs**

Set this property to "Button" to style the link as a button. The jQuery Mobile framework will enhance the link with markup and classes to style it as a button. Note that while the link will look like a button, it is still a link. Clicking on it will result in a GET request instead of a POST and will not automatically submit all fields in the form.

### **Default value**

Link - Adds no extra styling to the anchor.

### **Valid values**

'Link' or 'Button'

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **icon**

Specifies the jQuery Mobile icon to use with the button.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and will be ignored otherwise.

## **Default value**

Default - no icon is used

## **Valid values**

Any of the values listed in the property dropdown.

## **iconPosition**

Specifies the position of the button icon relative to the button text.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and an **icon** has been specified. It will be ignored otherwise.

### **Default value**

Blank - uses the jQuery Mobile default position of 'Left'.

### **Valid values**

Left - position the icon to the left of the button text.

Right - position the icon to the right of the button text.

Top - position the icon above the button text.

Bottom - position the icon below the button text.

No Text - draw the button with no text (icon only).

## **iconShadow**

Applies the theme shadow to the button's icon if set to true.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and an **icon** has been specified. It will be ignored otherwise.

### **Default value**

False – Don't apply a shadow

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **inline**

If set to true, this will make the button act like an inline button so the width is determined by the button's content. If set to false, the button width is the full width of its container, regardless of the content.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and will be ignored otherwise.

## **Default value**

False - the button is as wide as its container.

## **Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Note:** This property is only valid if the **displayAs** property is set to 'Button' and will be ignored otherwise.

### **Default value**

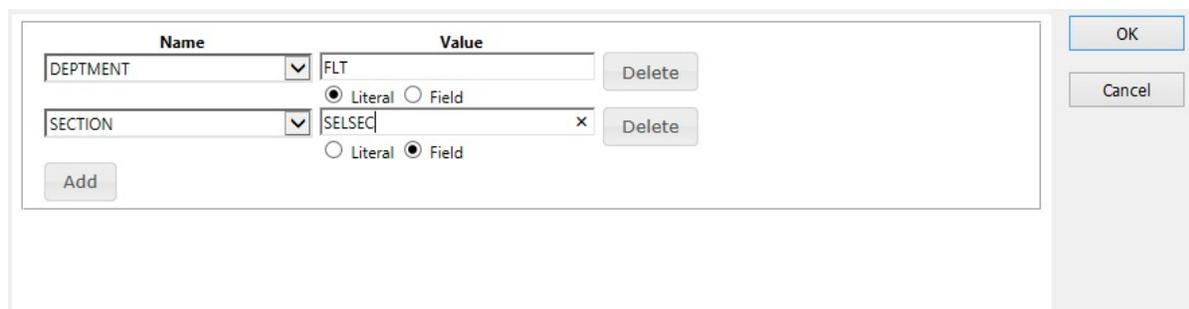
False - the standard size is used.

### **Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## onClickExtraFields

By default, the anchor generates a GET request with no field values submitted to the target webroutine. This property allows you to specify extra fields and values that should be sent to the target webroutine when the weblet is clicked. This property can only be set by using the custom property designer, invoked using the ellipses button in the property sheet.



The screenshot shows a dialog box for configuring the `onClickExtraFields` property. It features a table with two columns: "Name" and "Value".

Name	Value	
DEPARTMENT	FLT	Delete
SECTION	SELSEC	Delete

Below the table is an "Add" button. To the right of the table, there are two radio button options: "Literal" (selected) and "Field".

On the far right of the dialog box, there are "OK" and "Cancel" buttons.

This shows an output field in the current webroutine (`#SELSEC`) and a literal value ("FLT") being mapped to input fields (the "Name" column) in the target webroutine.

Note: You must set the `onClickWamName` and `onClickWrName` properties before editing this property so that the dropdown in the "Name" column can be correctly filled.

### Default Value

`document(")*/lxml:data/lxml:json[not(@id)]` (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## **onClickWamName**

Specifies the name of the WAM whose webroutine is executed when this weblet is clicked. (The webroutine name is specified in the onClickWrName property.)

This property is ignored if the **url** property is specified.

### **Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName).

### **Valid values**

The name of a WAM. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **onClickWrName**

Specifies the name of the webroutine that is executed when this weblet is clicked. (The name of the WAM containing the webroutine is specified in the onClickWamName property.)

This property is ignored if the **url** property is specified.

### **Default value**

No default value applies – either the url property or the on\_click\_wrname property must be specified.

### **Valid values**

The name of a Webroutine. The Webroutine must exist in the WAM specified in the onClickWamName property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## relationship

Tells jQuery mobile how it should load and display the target page. Possible values and their effects are:

- None     The page content is loaded using Ajax and replaces the current page content. The page is added to the browser history.
- Back     Clicking the link or button navigates to the previous page in the browser history (the same as clicking the back button in the browser). This option will cause any URL or Webroutine settings on the weblet to be ignored.
- Dialog    The page content is loaded using Ajax and it displayed in a dialog window. It is not added to the browser history.
- External   The entire page is loaded (that is, Ajax is not used) and added to the browser history. This has the same effect as setting **useAjax** to false but has a different semantic meaning, This setting should be used for links to another site of domain.

**Note:** For security reasons, links to another site will not use Ajax regardless of this setting.

### Default value

None.

### Valid values

None, Back, Dialog, External.

## **relativeImagePath (std\_anchor\_s1 only)**

The path and file name, relative to the images virtual directory, of the image to be displayed.

### **Default value**

Blank – no image is displayed.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

**shadow**

Applies the drop shadow style to the button if set to true.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## transition

Specifies the transition to use when displaying the next page. This property only applies if the next page is loaded with Ajax.

Transition effects that are shipped with the standard jQuery Mobile are:

- fade
- pop
- flip
- turn
- flow
- slidefade
- slide
- slideup
- slidedown,

See the [jQuery Mobile documentation](#) for information on creating custom transitions.

**Note:** Not all devices and browsers are able to support complex transitions. jQuery mobile will automatically fall back to a fade or no transition on such devices.

### Default value

slide

### Valid values

The name on any installed transitions or none

## **transitionDirection**

Specifies the direction of the transition animation.

**Note:** If the relationship property is set to "Back" the transition direction will be "reverse" regardless of this property value.

### **Default value**

forward

### **Valid values**

forward or reverse.

## **url**

Specifies a URL that will be loaded when the weblet is clicked.

If the `onClickWamName` **and** `onClickWrName` properties are both specified, this property will be ignored.

### **Default value**

# - equivalent to nothing.

### **Valid values**

Any valid URL string.

## **useAjax**

jQuery normally loads pages from the same domain using an Ajax request and then animates the page content into place when the load is complete. This usually results in faster loads (scripts and styles and not loaded again) and a more app-like appearance to you site.

Sometimes it might be necessary to force a complete reload of the page. This will most likely be when the target webrountine uses a different layout and needs to load different script and style resources.

Setting this property to false will force the page to be reloaded without an Ajax request.

See the "Linking Pages" section of the [jQuery Mobile documentation](#) for more information on how pages are loaded.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

## **9.4.2 Autocomplete (std\_autocomplete)**

The Autocomplete weblet provides suggestions while you type into the field. The suggestions are provided by a webroutine using Ajax.

## Properties – Autocomplete (std\_autocomplete)

The properties for this weblet are:

cache	Id	mini
corners	inset	name
extraFields	itemsSwatch	placeholder
fieldContainWrapper	label	sourceWamName
filterSwatch	labelField	sourceWrName
hideIf	listName	termField
hideLabel	minLength	valueField

## Id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**placeholder**

Specifies a short hint that describes what you are expected to type to get the suggestion list. The hint is displayed in the field when it is empty, and disappears when the field gets focus or contains a value (details vary by browser).

**Default value**

Blank

**Valid values**

Any string value.

## **minLength**

The minimum number of characters a user has to type before the Autocomplete activates. Use a sensible number according to the number of matching list items likely to be returned.

### **Default value**

1

### **Valid values**

Numeric value.

**sourceWamName**

The name of the WAM whose Webroutine provides the response data for this weblet.

**Default value**

The current WAM

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**sourceWrName**

The name of the Webroutine that provides the response data for this weblet.

**Default value**

Blank – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must be a JSON response weboutine and exist in the WAM specified in the sourceWamName property. A list of known JSON Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**termField**

The field name in the response handling webroutine that is to receive the current value in the autocomplete field.

**Default value**

None. Required field.

**Valid values**

Single quoted text field. Field must be in WEB\_MAP for \*INPUT in the response handling webroutine.

**listName**

The name of the list in the response webroutine to store the suggestion list.

**Default value**

Blank.

**Valid values**

Single quoted text . List must be in WEB\_MAP for \*OUTPUT in the response handling webroutine.

## **labelField**

The response data is a list, with either a label or value column or both. The label column is displayed in the suggestion menu. The value will be inserted into the hidden input element after the user selected something from the menu. If just one column is specified, it will be used for both, eg. if you provide only value-properties, the value will also be used as the label.

## **Default value**

Blank.

## **Valid values**

Single quoted text. Column name but exist in list returned by the response handling webroutine.

## **valueField**

The response data is a list, with either a label or value column or both. The label column is displayed in the suggestion menu. The value will be inserted into the hidden input element after the user selected something from the menu. If just one column is specified, it will be used for both, eg. if you provide only value-properties, the value will also be used as the label.

## **Default value**

Blank.

## **Valid values**

Single quoted text. Column name but exist in list returned by the response handling webroutine.

## extraFields

This property allows you to specify extra fields and values that should be sent to the response webroutine when a request for a suggestion list is made.

This property can only be set by using the custom property designer, invoked using the ellipses button in the property sheet.

Name	Value	
DEPTMENT	FLT	Delete
	<input checked="" type="radio"/> Literal <input type="radio"/> Field	
SECTION	#SELSEC	Delete
	<input type="radio"/> Literal <input checked="" type="radio"/> Field	

Add

OK

Cancel

This shows an output field in the current webroutine (`#SELSEC`) and a literal value ("FLT") being mapped to input fields (the "Name" column) in the response webroutine.

Note: You must set the `sourceWamName` and `sourceWrName` properties before editing this property so that the dropdown in the "Name" column can be correctly filled.

## Default Value

`document(")/*//lxml:data/lxml:json[not(@id)]` (this indicates no items have been defined for this weblet).

## Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

**cache**

Set to true to save the server response locally and reuse if the term is entered again later in the same field.

**Default value**

False.

**Valid values**

true(), false() or a valid expression.

**corners**

Specifies if the search field should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **filterSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the filter input field.

### **Default value**

Default - Inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **itemsSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the list items of the suggestion list

### **Default value**

Default - Inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **inset**

If lists are embedded in a page with other types of content, an inset list packages the list into a block that sits inside the content area with a bit of margin and rounded corners (theme controlled). Setting **inset** to True applies the inset appearance.

Note: all standard, non-inset lists have a -15 pixel margin to negate the 15 pixels of padding on the content area to make lists fill to the edges of the screen. If you add other content above or below a list, the negative margin may make these elements overlap so you'll need to add additional spacing in your custom CSS.

## **Default value**

False

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

### **9.4.3 Boolean (std\_boolean)**

The Boolean weblet represents boolean values. The weblet generates an `<input type="checkbox">` element in the HTML source which jQuery Mobile then converts into a sliding switch.

The Boolean weblet is the default weblet for boolean fields but it can also be used with any field that has only two valid values (for example: Sex) by configuring the `trueValue` and `falseValue` properties appropriately.

## Properties - Boolean (std\_boolean)

This weblet's properties are:

autofocus	form	rdmlxDatatype
class	hideIf	style
corners	hideLabel	swatch
disabled	id	tabindex
displayMode	label	title
falseDisplay	mini	trueDisplay
falseValue	name	trueValue
fieldContainWrapper		value

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown. The checked state of the weblet is determined by comparing the value with the trueValue property.

**Default value**

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the checkbox. Any value that does not match the trueValue property is treated as false.

**Valid values**

A string value or the name of a field, system variable or multilingual text variable.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**corners**

Specifies if the flipswitch should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**displayMode**

Controls whether the weblet accepts input or is output only.

**Default value**

Blank (equivalent to 'input').

**Valid values**

input or output.

**falseDisplay**

The text to display in the switch when it is in the off position.

**Default value**

No

**Valid values**

Any string value.

## **falseValue**

The field value that represents the false or off state. This is the value that is send to the server when the field is submitted, it is not necessarily the value that will set the weblet to the off state. Instead, the off state is set if the field value is not equal to the **trueValue** property.

## **Default value**

false

## **Valid value**

Any string value.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **rdmlxDataType**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

## **Valid values**

Any valid HTML attribute string.

**trueDisplay**

The text to display in the switch when it is in the on position.

**Default value**

Yes

**Valid values**

Any string value.

**trueValue**

The field value that represents the true or on state. This is the value that is send to the server when the field is submitted. The weblet will be initially set to the on state if the **value** field contains this value.

**Default value**

true

**Valid value**

Any string value.

#### 9.4.4 Button (std\_button\_s1 and std\_button\_v2)

The button weblet represents an HTML <button> element which creates a clickable button. Unlike a button created with the <input> element (Input box weblet) the button weblet can contain custom HTML like formatted text and images. Weblet std\_button\_v2 has the same properties as std\_button\_s1, but with the difference that you can add content to the button element.

There are three possible types of button:

**Submit** A 'submit' button calls a web routine sending the values of all named fields in the parent form (or the form specified in the form property). You should use a submit button whenever sending data to a web routine that results in a database update or when you require HTML 5 validation to be performed on the fields before the form is submitted.

**Reset** A 'reset' button resets all fields in the parent <form> (or the form specified in the form property) to the value they had when the page was loaded.

**Button** A 'button' button has no default action when clicked. You can attach a custom JavaScript function to a button by using the presubmitJS property.

Use std_button_v2 only if you need to add content to the button element. For all other cases, use std_button_s1.
--

## Properties - Button (std\_button\_s1 and std\_button\_v2)

This Button weblet's properties are:

autofocus	icon	presubmitJS
caption	iconPosition	shadow
class	iconShadow	style
corners	id	swatch
disabled	inline	tabindex
form	internal_id	title
formaction	mini	transition
formenctype	name	transitionDirection
formmethod	onClickExtraFields	type
formnovalidate	onClickWamName	useAjax
formtarget	onClickWrName	value
hideIf		

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

The value of the button. This value is not displayed (use the **caption** property for that). It is the value that is submitted to the target webroutine when the button is clicked.

**Default value**

Blank

**Valid values**

Any string value.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**caption**

The caption for the button. This is the default content for the button. Custom content added to the button may overwrite this value.

**Default value**

Blank

**Valid values**

Any string value.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**corners**

Specifies if the button should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

## **formation**

A URL that specifies where to send the form-data when a form is submitted. Only valid for **type**="submit". Normally, the LANSAs runtime framework takes care of working out the correct URL based on the onClickWrName and onClickWR fields. Setting this property will override this default behaviour.

### **Default Value**

Blank

### **Valid values**

Any valid URL.

## **formenctype**

Specifies the encoding type to use when sending the form. Only valid for **type**="submit". Normally you will not need to set this as the default type is fine for sending most forms to LANSAs servers. If you are sending a form to a non-LANSAs server then specific requirements of that server may require a different encoding type.

### **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of application/x-www-form-urlencoded.

### **Valid values**

application/x-www-form-urlencoded	All characters are encoded before sent (spaces are converted to "+" symbols, and special characters are converted to ASCII HEX values).
multipart/form-data	No characters are encoded. This value is required when you are using forms that have a file upload control.
text/plain	Spaces are converted to "+" symbols, but no special characters are encoded.

## **formmethod**

Specifies which HTTP method to use to send the form data. Only valid for **type="submit"**.

## **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of POST.

## **Valid values**

'GET' or 'POST'.

## **formnovalidate**

Specifies that the form should not be validated prior to submission. Only valid for **type**="submit". Setting this property to True will suppress any validation that has been turned on in the parent <form> element, setting it to False will not turn on validation that has been turned off in the parent <form> element.

For standard shipped layouts the <form> validation is configured with the validationMethod property of the layout weblet.

### **Default value**

False.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **formtarget**

Specifies the frame or window in which to display the response after submitting the form. Only valid for **type="submit"**.

This property is part of the HTML 5 specification and is provided for completeness. Care should be taken with its use. Different mobile browsers have different levels of support for frames and windows and may handle this property differently. It may also cause conflicts with jQuery Mobile's Ajax mechanisms.

### **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of blank also (display in the current page).

### **Valid values**

\_blank, \_self, \_parent, \_top, framename.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**icon**

Specifies the jQuery Mobile icon to use with the button.

**Default value**

Default - no icon is used

**Valid values**

Any of the values listed in the property dropdown.

## **iconPosition**

Specifies the position of the button icon relative to the button text.

### **Default value**

Blank - uses the jQuery Mobile default position of 'Left'.

### **Valid values**

- Left      Position the icon to the left of the button text.
- Right     Position the icon to the right of the button text.
- Top       Position the icon above the button text.
- Bottom    Position the icon below the button text.
- No Text   Draw the button with no text (icon only).

## **iconShadow**

Applies the theme shadow to the button's icon if set to true.

### **Default value**

False – Don't apply a shadow

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**inline**

If set to true, this will make the button act like an inline button so the width is determined by the button's content. If set to false, the button width is the full width of its container, regardless of the content.

**Default value**

False - the button is as wide as its container.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False –Use the standard size.

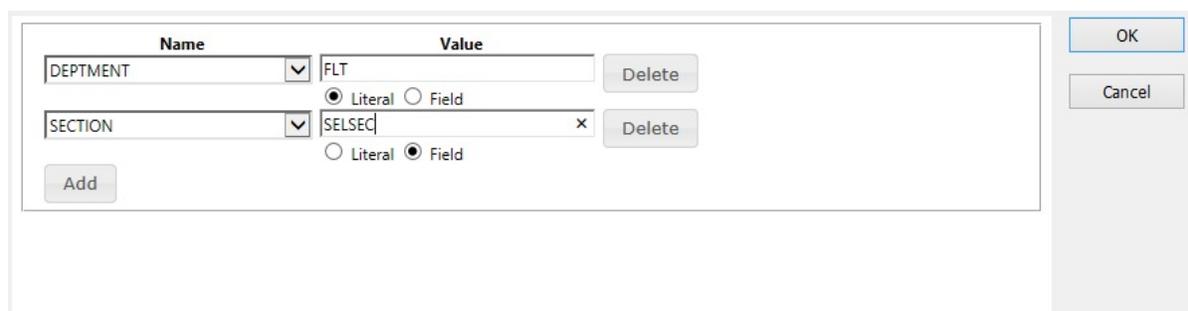
**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## onClickExtraFields

By default, the button generates a POST request and automatically submits all fields in its parent <form> to the target webroutine. This property allows you to specify extra fields and values that should be sent to the target webroutine when the weblet is clicked.

This property can only be set by using the custom property designer, invoked using the ellipses button in the property sheet.



The screenshot shows a dialog box for configuring the `onClickExtraFields` property. It features a table with two columns: "Name" and "Value".

Name	Value	
DEPTMENT	FLT	Delete
	<input checked="" type="radio"/> Literal <input type="radio"/> Field	
SECTION	SELSEC	Delete
	<input type="radio"/> Literal <input checked="" type="radio"/> Field	

Below the table is an "Add" button. To the right of the table are "OK" and "Cancel" buttons.

This shows an output field in the current webroutine (`#SELSEC`) and a literal value ("FLT") being mapped to input fields (the "Name" column) in the target webroutine.

Note: You must set the `onClickWamName` and `onClickWrName` properties before editing this property so that the dropdown in the "Name" column can be correctly filled.

### Default Value

`document(")/*/lxml:data/lxml:json[not(@id)]` (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## **onClickWamName**

Specifies the name of the WAM whose webroutine is executed when this weblet is clicked. (The webroutine name is specified in the onClickWrName property.)

This property has no effect if the **type** property is set to "button" or "reset".

### **Default value**

If not specified, the current WAM is used. (\$lweb\_WAMName).

### **Valid values**

The name of a WAM. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **onClickWrName**

Specifies the name of the webroutine that is executed when this weblet is clicked. (The name of the WAM containing the webroutine is specified in the onClickWamName property.)

This property has no effect if the **type** property is set to "button" or "reset".

### **Default value**

No default value applies – either the url property or the on\_click\_wrname property must be specified.

### **Valid values**

The name of a Webroutine. The Webroutine must exist in the WAM specified in the onClickWamName property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **presubmitJS**

JavaScript code to be run prior to the submission of the form.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript function, or JavaScript code followed by a semicolon (;).

If you want to execute the presubmit JavaScript only, without running the JavaScript that submits the request, append **return false;** to your presubmit JavaScript.

**shadow**

Applies the drop shadow style to the button if set to true.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

## **Valid values**

Any valid HTML attribute string.

## transition

Specifies the transition to use when displaying the next page. This property only applies if the next page is loaded with Ajax.

Transition effects that are shipped with the standard jQuery Mobile are:

- fade
- pop
- flip
- turn
- flow
- slidefade
- slide
- slideup
- slidedown,

See the [jQuery Mobile documentation](#) for information on creating custom transitions.

**Note:** Not all devices and browsers are able to support complex transitions. jQuery mobile will automatically fall back to a fade or no transition on such devices.

### Default value

slide

### Valid values

The name on any installed transitions or none

## **transitionDirection**

Specifies the direction of the transition animation.

**Note:** If the relationship property is set to "Back" the transition direction will be "reverse" regardless of this property value.

### **Default value**

forward

### **Valid values**

forward or reverse.

## **type**

Specifies the type of the button. There are three possible button types:

**Submit** Tapping a submit button causes the <form> that the button belongs to to be sent to the server.

**Reset** Tapping a reset button causes the browser to reset all field in the form to the initial values they contained when the page was loaded.

**Button** This button has no default action when clicked. You must use JavaScript to define custom behavior.

## **Default value**

submit

## **Valid values**

submit, reset, button

## **useAjax**

jQuery normally loads pages from the same domain using an Ajax request and then animates the page content into place when the load is complete. This usually results in faster loads (scripts and styles and not loaded again) and a more app-like appearance to you site.

Sometimes it might be necessary to force a complete reload of the page. This will most likely be when the target webrountine uses a different layout and needs to load different script and style resources.

Setting this property to false will force the page to be reloaded without an Ajax request.

See the "Linking Pages" section of the [jQuery Mobile documentation](#) for more information on how pages are loaded.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### **9.4.5 Checkbox (std\_checkbox)**

This weblet creates a standard checkbox control which jQuery Mobile will modify to fit the current theme. Note that a checkbox field is only submitted to a web routine if the checkbox is checked. If the checkbox is not checked, no value is submitted and the field in the web routine will contain its default value.

## Properties - Checkbox (std\_checkbox)

This weblet's properties are:

autofocus	label	selectedValue
disabled	mini	swatch
form	name	tabindex
hideIf	rdmlxDatatype	title
id	required	value

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

The initial value of the checkbox. This value is compared with the selectedValue property to determine if the checkbox should be initially checked.

**Default value**

Blank

**Valid values**

Any string value.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field.

**Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

**Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **rdmlxDataType**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

## **required**

Specifies that the checkbox must be checked before the form can be submitted. Note that, at the time of writing, Safari and Internet Explorer do not support this property.

**Note:** HTML 5 form validation can be turned off by adding a `novalidate` attribute to the `<form>` tag or submit button (this is automatically done by setting `validationMethod` to 'none' on the standard shipped layouts). In addition, different browsers have differing levels of support for HTML 5 validation and there are many ways for malicious users to bypass client-side validation. So, while client-side validation can improve the user experience, you should never rely on it. Always back it up with server-side validation.

## **Default value**

False

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**selectedValue**

Specifies the field value that represents the selected state. When the checkbox is initialized, the actual value is compared with this value (using a case sensitive comparison) to determine if the checkbox is checked. When the form is submitted, this is the value that is sent if the checkbox is checked. No value is sent for an unchecked checkbox.

**Default value**

true

**Valid values**

Any string value.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

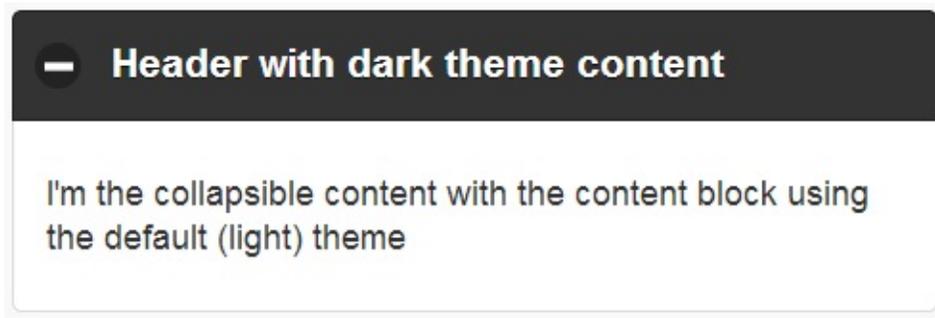
Blank

## **Valid values**

Any valid HTML attribute string.

## 9.4.6 Collapsible Block (std\_collapsible)

A Collapsible Block is a section of content that the user can show or hide by clicking on its header.



An expanded Collapsible Block

## Properties - Collapsible Block (std\_collapsible)

This weblet's properties are:

collapseCueText headerLevel id  
collapsed headerSwatch inset  
contentSwatch headerText internal\_id  
expandCueText hideIf

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**collapsed**

Specifies the initial state of the weblet.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **contentSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the content area of the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **headerLevel**

The header section of the weblet is rendered inside an HTML header tag. This property specifies the level of that header (h1, h2, h3, h4, h5, h6, h7 or h8). The header level has no effect of the visual appearance of the weblet but may be important for the semantic structure of your document. This may be important to how other systems such as search engines or assistive technologies process your page.

### **Default value**

3

### **Valid values**

1, 2, 3, 4, 5, 6, 7 or 8.

## **headerSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the header area of the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**headerText**

Specifies the text to display in the header area.

**Default value**

Collapsible Block

**Valid values**

Any string value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **inset**

By default collapsible blocks have an inset appearance. To make them full width without corner styling set this property to false.

Default value

True – the block will have a margin.

Valid values

True, False, or any valid XPath expression that returns a boolean value.

## **collapseCueText**

The text used to provide audible feedback for users with screen reader software.

### **Default value**

"click to collapse contents"

### **Valid values**

Any string value.

## **expandCueText**

The text used to provide audible feedback for users with screen reader software.

### **Default value**

"click to expand contents"

### **Valid values**

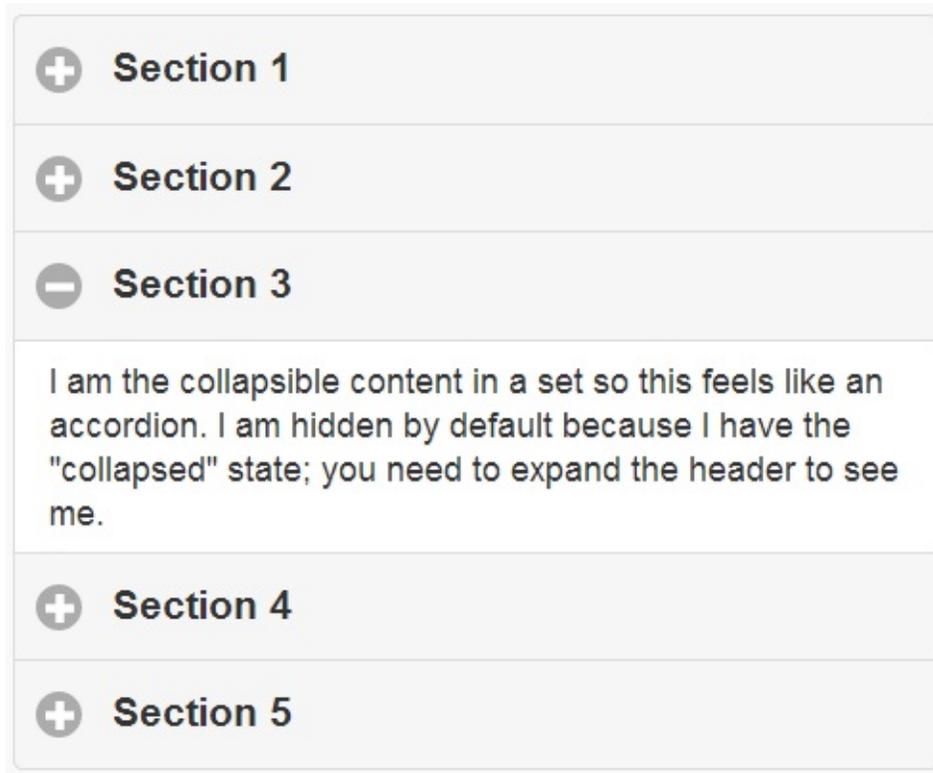
Any string value.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

## 9.4.7 Collapsible Set (std\_collapsibleset)

The Collapsible Set weblet is a wrapper that can be placed around multiple Collapsible Block weblets to create an accordion. When one Collapsible Block is expanded, the others in the set are collapsed. This weblet does not create the Collapsible Block weblets. You must create those and place them into the content area of the Collapsible Set weblet.



## **Properties - Collapsible Set (std\_collapsible)**

This weblet's properties are:

[hideIf](#)

[id](#)

[internal\\_id](#)

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

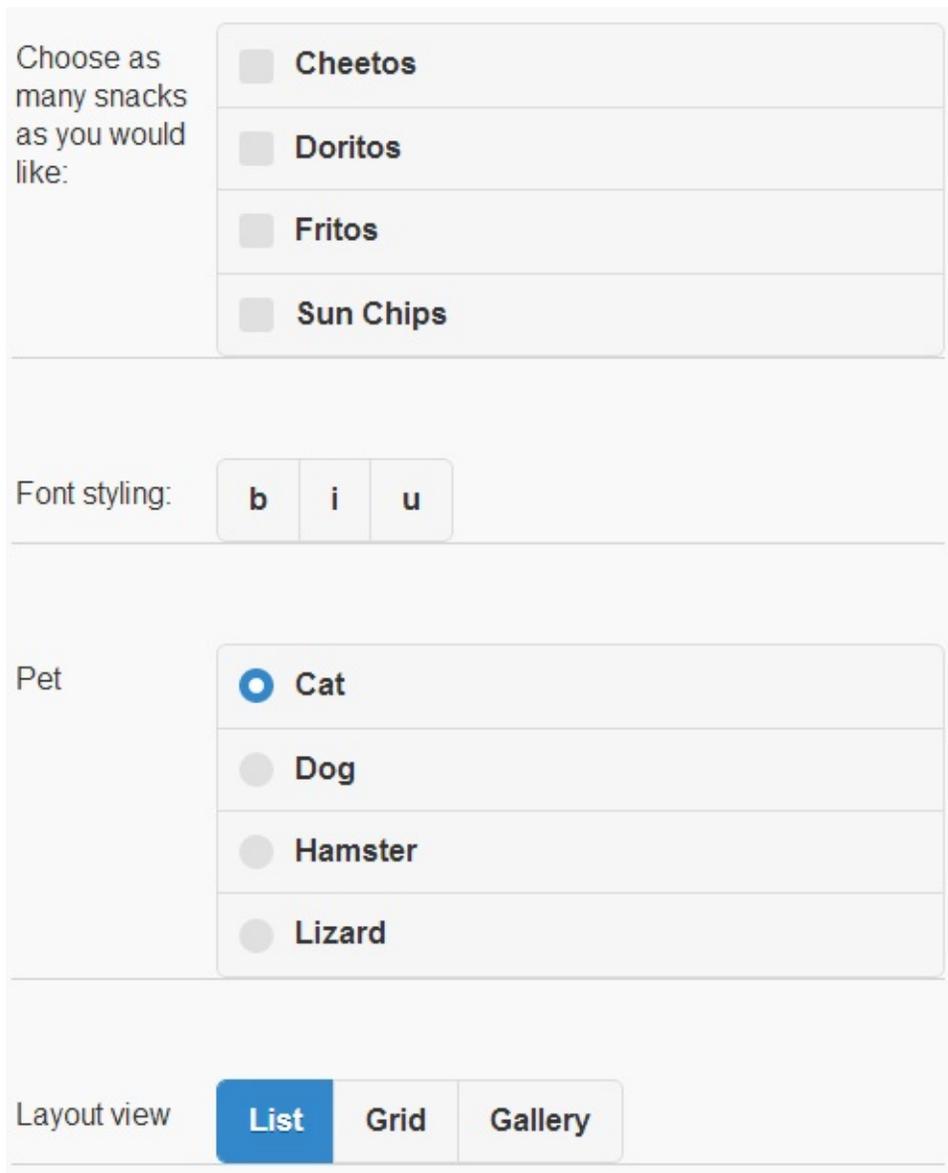
True, False, or any valid XPath expression that returns a boolean value.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

## 9.4.8 Control Group (std\_controlgroup)

The Control Group weblet is a wrapper that can be placed around multiple buttons, checkboxes or radio buttons to group them together. The framework will automatically remove all margins between buttons and round only the top and bottom corners of the set.



The image displays four examples of grouped controls. Each example consists of a label on the left and a group of controls on the right, all enclosed in a rounded rectangular container.

- Example 1:** Label: "Choose as many snacks as you would like:". Controls: Four checkboxes, each followed by a snack name: "Cheetos", "Doritos", "Fritos", and "Sun Chips".
- Example 2:** Label: "Font styling:". Controls: Three buttons labeled "b", "i", and "u".
- Example 3:** Label: "Pet". Controls: Four radio buttons, each followed by a pet name: "Cat", "Dog", "Hamster", and "Lizard". The "Cat" radio button is selected.
- Example 4:** Label: "Layout view". Controls: Three buttons labeled "List", "Grid", and "Gallery". The "List" button is highlighted in blue.

Examples of grouped controls

## Properties - Control Group (std\_controlgroup)

This weblet's properties are:

<a href="#">class</a>	<a href="#">id</a>	<a href="#">orientation</a>
<a href="#">fieldContainWrapper</a>	<a href="#">internal_id</a>	<a href="#">style</a>
<a href="#">hideIf</a>	<a href="#">label</a>	<a href="#">swatch</a>
<a href="#">hideLabel</a>	<a href="#">mini</a>	

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label. It is currently not possible to hide the label for this weblet so that it is still accessible to assistive technologies (as the **hideLabel** property will do on other weblets) so setting this property to true is the same as setting the **label** property to a blank value. This property exists for consistency with other weblets and so that, should future browsers make it possible to hide the label accessibly, that will automatically happen without you needing to change anything.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

**orientation**

Specifies how the controls in the group should be arranged.

**Default value**

vertical

**Valid values**

horizontal or vertical.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### 9.4.9 Select Menu (`std_dropdown`)

The select menu is based on a native `select` element, which is hidden from view and replaced with a custom-styled select button that matches the look and feel of the jQuery Mobile framework.

By default, the framework leverages the native OS options menu to use with the custom button. When the button is clicked, the native OS menu will open. When a value is selected and the menu closes, the custom button's text is updated to match the selected value.

The Select Menu weblet also offers the option of generating custom menus instead of the native OS menu. The custom menu supports disabled options, multiple selection (whereas native mobile OS support for both is inconsistent) and adds an elegant way to handle placeholder values. See the [useNativeControl](#) property for details.

## Properties - Select Menu (std\_dropdown)

This weblet's properties are:

addErrorDiv	id	selectorValueField
autofocus	inline	shadow
class	items	style
corners	label	swatch
disabled	mini	tabindex
displayMode	multiple	title
fieldContainWrapper	multiSelectCodeField	updateFieldsToSubmit
form	multiSelectListname	updateOnFieldChange
hideIf	name	updateProtocol
hideLabel	overlaySwatch	updateWamName
icon	placeholder	updateWrName
iconPosition	rdmlxDatatype	useNativeControl
iconShadow	required	value

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**value**

Specifies the initial value of the weblet. This value will be compared with the code value of each item to determine which item to initially select.

**Default value**

Blank

**Valid values**

Any string value.

## **addErrorDiv**

When set to True, a <div> element will be added just after the weblet to display validation errors. the <div> will be hidden until a validation error occurs and will be hidden again when the error is cleared.

When a validation method has been set the error <div> will reserve some space for itself on the page even when it is hidden to avoid confusing rearrangements of the screen when the error is displayed. If you are using HTML 5 validation on the form but not on this field then you may want to set this property to False to reclaim that space.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**corners**

Specifies if the button should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**displayMode**

Controls whether the weblet accepts input or is output only.

**Default value**

Blank (equivalent to 'input').

**Valid values**

input or output.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**icon**

Specifies the jQuery Mobile icon to use with the button.

**Default value**

Default - a down arrow icon is used

**Valid values**

Any of the values listed in the property dropdown.

## **iconPosition**

Specifies the position of the button icon relative to the button text.

### **Default value**

Blank - uses the jQuery Mobile default position of 'Left'.

### **Valid values**

Left - position the icon to the left of the button text.

Right - position the icon to the right of the button text.

Top - position the icon above the button text.

Bottom - position the icon below the button text.

No Text - draw the button with no text (icon only).

## **iconShadow**

Applies the theme shadow to the button's icon if set to true.

### **Default value**

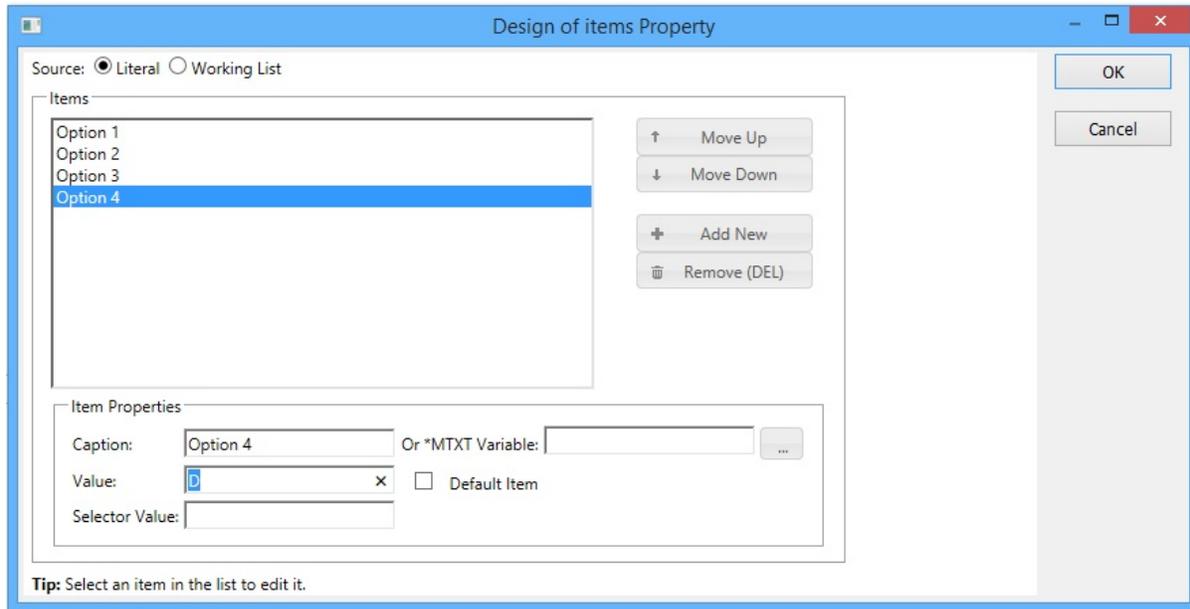
False – Don't apply a shadow

### **Valid values**

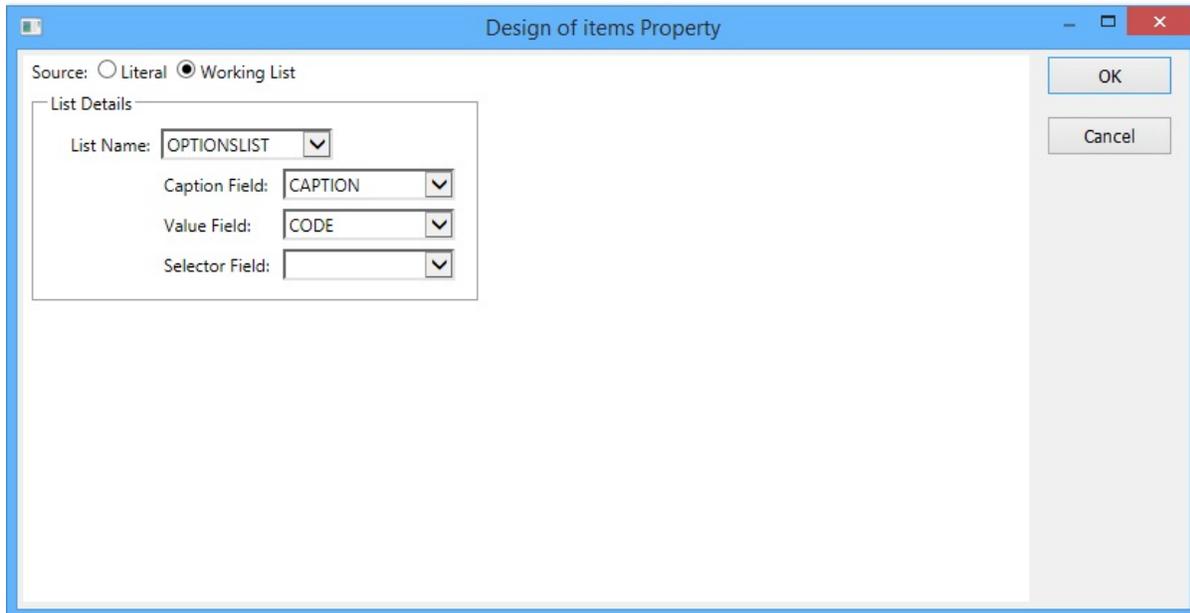
True, False, or any valid XPath expression that returns a boolean value.

## items

An XML nodeset specifying the items to appear in the weblet. This can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. The property designer can be used to specify a hard coded set of items or the name of a working list to get the items from. When using a working list, the list must be specified as \*JSON in the output Web Map.



This graphic shows a list configured with 4 items. Check(select) the Default Item check box for the item that is to be selected if no value is preselected. The Selector value can be used to filter the list down to a smaller set of displayed values at runtime.



This graphic shows the items property editor configured to use a working list. The Selector field can be used to filter the list down to a smaller set of displayed values at runtime.

### **Default value**

document(")/\*/lxml:data/lxml:dropdown (this indicates no items have been defined for this weblet.)

### **Valid values**

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

**inline**

If set to true, this will make the button act like an inline button so the width is determined by the button's content. If set to false, the button width is the full width of its container, regardless of the content.

**Default value**

False - the button is as wide as its container.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **multiple**

A Boolean property that controls whether multiple selections are allowed in the list box. If multiple selections are allowed, the `multiSelectListname` and `multiSelectCodefield` properties must be specified.

Note that mobile browser support for multiple selections is inconsistent. To avoid browser compatibility issues, you may want to set **useNativeControl** to false and let jQuery Mobile render a custom interface for the list.

### **Default value**

False

### **Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **multiSelectCodeField**

The name of the field in the **multiSelectListname** working list that holds the code value of the selected menu items.

### **Default value**

Blank.

### **Valid values**

The name of a field. Click the corresponding dropdown button in the property sheet to choose from a list of known fields.

## **multiSelectListname**

The working list that contains the selected entries for the menu. The working list should contain only the code field that is specified in the **multiSelectCodeField** property. If the **multiple** property is false, this property is ignored.

### **Default value**

Blank.

### **Valid values**

The name of a working list. Click the corresponding dropdown button in the property sheet to choose from a list of known working lists.

## **overlaySwatch**

Specifies the default jQuery Mobile theme swatch, or color scheme, to apply to the overlay layer for the dialog-based custom select menus and the outer border of the smaller custom menus.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **placeholder**

Specifies a short hint that describes the expected value of the input field (for example, a sample value or a short description of the expected format). The hint is displayed in the field when it is empty, and disappears when the field gets focus or contains a value (details vary by browser).

## **Default value**

Blank

## **Valid values**

Any string value.

## **rdmlxDataType**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

## **required**

Specifies that a value must be entered in this weblet before the form can be submitted. . Note that, at the time of writing, Safari and Internet Explorer do not support this property.

**Note:** HTML 5 form validation can be turned off by adding a novalidate attribute to the <form> tag or submit button (this is automatically done by setting validationMethod to 'none' on the standard shipped layouts). In addition, different browsers have differing levels of support for HTML 5 validation and there are many ways for malicious users to bypass client-side validation. So, while client-side validation can improve the user experience, you should never rely on it. Always back it up with server-side validation.

## **Default value**

False

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **selectorValueField**

The name of a field whose value is used to filter the list supplied to the weblet into a smaller list for display. When building the display list, this value is compared with the value in the lists "selector" column. If a match is found the entry is included in the displayed list.

This can be useful for reducing the work done at the server. Instead of calculating the list entries every time it is executed, the webroutine could output a pre-built list with all possible values and a selector value. The browser can then reduce the list to a subset based on the selector value. Keep in mind, however, that the reduction in server load should be balanced against an increase in network bandwidth used sending more data to the browser.

This can also be used to allow a dynamic list to refresh without having to make a server request. If the field being monitored for updates is also the selectorValueField then the weblet can rebuild itself by applying the new selector value to the list initially passed to it.

### **Default value**

Blank. No filtering is done.

### **Valid values**

The name of any output field in the current Webroutine

**shadow**

Applies the drop shadow style to the button if set to true.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

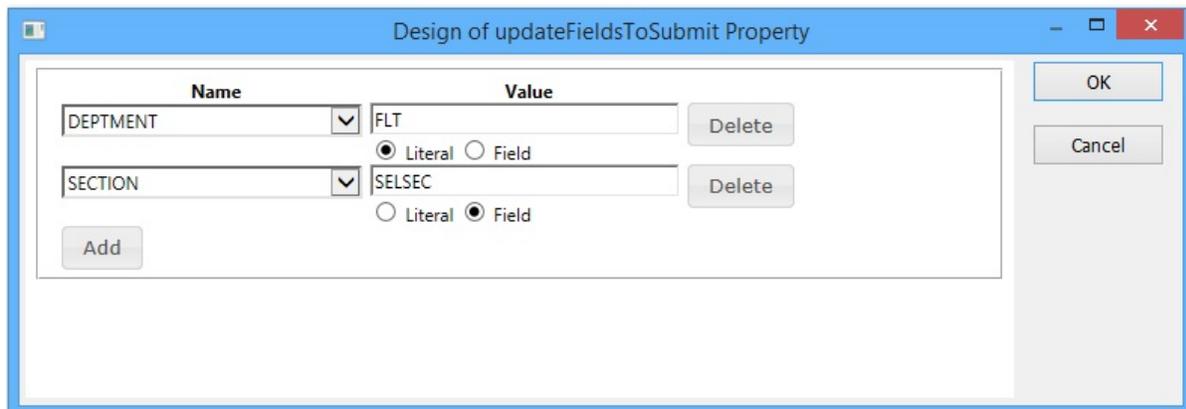
## **Valid values**

Any valid HTML attribute string.

## updateFieldsToSubmit

Specifies the fields and values to submit when calling the "update" webroutine to refresh the list values.

This property can only be set by using the custom property designer, invoked using the ellipses button in the property sheet.



This shows an output field in the current webroutine (#SELSEC) and a literal value ("FLT") being mapped to input fields (the "Name" column) in the target webroutine.

Note: You must set the updateWamName and updateWrName properties before editing this property so that the dropdown in the "Name" column can be correctly filled.

### Default Value

document(")\*/lxml:data/lxml:json[not(@id)] (this indicates no items have been defined for this weblet).

### Valid values

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## **updateOnFieldChange**

The ID of a field to monitor for changes. If a change occurs in the monitored field the select box will refresh. If **updateWamName** and **updateWrName** have been specified, the weblet will call the Webroutine to request a fresh copy of the working list, if one was specified in the items property. Otherwise it will re-apply the **selectorValueField** filter to the list it already has and rebuild the menu list from that.

Fields built around the <input> and <select> elements can be monitored (this is all standard fields except the Text Area).

### **Default value**

Blank - no field is monitored.

### **Valid values**

The ID of any <input> or <select> element on the current page. For standard weblets, this is the same as the **id** property of the weblet.

## **updateProtocol**

The HTTP protocol that should be used when calling the update webroutine.

### **Default value**

Blank.- Uses the same protocol that the current page used.

### **Valid values**

http: or https:

## **updateWamName**

The name of the WAM to be invoked when refreshing the list.

### **Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

### **Valid values**

The name of a WAM. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **updateWrName**

The name of the Webroutine to be invoked when refreshing the list. This webroutine must be defined as \*JSON.

### **Default value**

Blank.

### **Valid values**

The name of a Webroutine. The Webroutine must exist in the WAM specified in the updateWamName property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **useNativeControl**

Specifies whether the browser should display the select menu using its own UI or if jQuery Mobile should provide the UI. Letting the browser display its own UI provides a user experience that is specific to the device and familiar to the user. Letting jQuery Mobile display the UI provides a user experience that is themeable and consistent across all devices.

Support for multiple selections is inconsistent among different browsers so, if your menu supports multiple selections, you may need to set this property to False.

Keep in mind that there is overhead involved in parsing the native select to build a custom menu. If there are a lot of selects on a page, or a select has a long list of options, this can impact the performance of the page, so we recommend using custom menus sparingly.

### **Default value**

True

### **Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## 9.4.10 File Upload (std\_fileupload)

### Properties – File Upload

The file upload weblet allows you to select files to upload to the application server (into a temporary directory). The webroutine that receives the file upload can then manipulate the uploaded files as required.

Use the context menu to create an skeleton for the Ajax webroutine for handling the file upload.

## Properties – File Upload

caption	id	successCallback
class	inline	tabindex
disabled	MaxFileSize	uploadWamName
failCallback	MaxNumberOfFiles	uploadWrName
hideIf	name	

**name**

The name of LOB field to receive the uploaded file temporary path.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

**id**

A unique id for the weblet. The default is the same as the name property and normally you would leave it as that. In some special circumstances you may have multiple weblets, in multiple forms, visualizing the same field. In those cases you would need to set this property to give each one a unique ID.

**Default value**

\$name The same as the name property

**Valid values**

Any string starting with a letter ([A-Za-z]) followed by any number of letters, digits([0-9]), hyphens ("-") or underscores ("\_").

**caption**

Specifies the caption for the button to add files.

**Default value**

Default: "Select files"

**Valid values**

Any string value.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**inline**

If set to true, this will make the file upload button act like an inline button so the width is determined by the button's content. If set to false, the button width is the full width of its container, regardless of the content.

**Default value**

False - the button is as wide as its container.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **uploadWamName**

The name of the WAM whose Webroutine receives the file uploaded by this weblet.

### **Default value**

The current WAM

### **Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

## **uploadWrName**

The name of the Webroutine that receives the file uploaded by this weblet. It must be a JSON webroutine. Its response is passed to the optional JavaScript callback functions to provide feedback to the user.

### **Default value**

Blank – a Webroutine name must be specified.

### **Valid values**

The name of a Webroutine in single quotes. The Webroutine must be a JSON response weboutine and exist in the WAM specified in the uploadWamName property. A list of known JSON Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

## **MaxFileSize**

The maximum file size allowed in Megabytes.

### **Default value**

5 – 5Mb

### **Valid values**

An integer value. Must be consistent with maximum value for file uploads defined in the application server. Note: Browsers may have their own limits.

## **MaxNumberOfFiles**

The maximum number of files allowed.

### **Default value**

1

### **Valid values**

An integer value.

## **successCallback**

The name of the optional JavaScript function to call when the file is successfully uploaded. The function is called with two arguments: The event object (null if not available) and the JSON webroutine response from the file upload webroutine.

### **Default Value**

None

### **Valid values**

The name of a JavaScript function.

## **failCallback**

The name of the optional JavaScript function to call when the file upload fails. The function is called with two arguments: The event object (null if not available) and a constructed JSON webroutine response with the error messages (mimics messages issued by a webroutine).

## **Default Value**

None

## **Valid values**

The name of a JavaScript function.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled. If the browser doesn't support ActiveX, the weblet is automatically disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or any valid expression that returns True or False.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

### 9.4.11 Footer (std\_footer)

A footer bar is a jQuery Mobile toolbar at the bottom of the page. A footer can be positioned at the bottom of the page or "fixed" to the bottom of the screen (so it remains in position when the page is scrolled). A footer can also be configured to show or hide itself when the user taps on the screen.

**Note:** The standard shipped layouts will provide a footer for you. The footer weblet is only required when creating a custom layout, or using a layout that does not provide one.

## Properties - Footer (std\_footer)s

This weblet's properties are:

fullscreenMode id persistentFooterId  
hideIf internal\_id position  
swatch

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **fullscreenMode**

A fullscreen footer is a footer with a position of "fixed" except that the footer overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the footer in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **position** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **persistentFooterId**

In situations where the footer is a global navigation element, you may want it to appear fixed so it doesn't scroll out of view. It's also possible to make a fixed footer persistent so it appears to not move between page transitions. This can be accomplished by using the persistent footer feature included in jQuery Mobile.

To make a footer persistent between transitions, assign an id value to the **persistentFooterId** of all relevant pages and use the same id value for each. For example, by setting **persistentFooterId** to "myfooter" to the current page and the target page, the framework will keep the footer anchors in the same spot during the page animation. This effect will only work correctly if the footers are set to position="fixed" so they are in view during the transition.

### **Default value**

Blank - the footer is not persistent.

### **Valid values**

Any string value.

## **position**

Footers can be positioned on the page in a few different ways. By default, the footer uses the "inline" positioning mode. In this mode, the footer sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the footer to the bottom of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the footer will fall back to static, inline position in the page.

## **Default value**

Inline

## **Valid values**

inline or fixed.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

## 9.4.12 Layout Grid (std\_gridlayout)

Using multiple column layouts isn't generally recommended on a mobile device because of the narrow screen width, but there are times where you may need to place small elements side-by-side (like buttons or navigation tabs, for example).

The Layout Grid weblet provides a simple way to build between 2 to 5 CSS-based columns.

Grids are 100% width, completely invisible (no borders or backgrounds) and don't have padding or margins, so they shouldn't interfere with the styles of elements placed inside them.

The Layout Grid weblet is designed to repeat the same content in each cell by iterating over an RDMLX working list. If you want to create a custom grid with different content in each cell, you should create the necessary HTML directly. You can find more details in the "Content Formatting" section of the [jQuery Mobile documentation](#).

## **Properties - Layout Grid (std\_gridlayout)**

This weblet's properties are:

`columns` `internal_id` `isOutputOnly` `listname`

**columns**

The number of columns in the grid.

**Default value**

2

**Valid values**

2 - 5

## **isOutputOnly**

Indicates that the list specified by the listname property is only being used by this weblet for output. When a list is being used for input, the LANSAs framework needs to create some hidden fields and do some extra processing to make sure the list data is sent to the server correctly. If the list is being used by this weblet for output purposes only, you can slightly improve performance and reduce the risk of conflicts with other weblets using the same list if you set this property to True.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**listname**

The name of the RDMLX working list to iterate over. One cell will be created for each row in the list. If no list is specified then a single row of cells will be created.

**Default value**

Blank

**Valid values**

The name of a working list output by the current webroutine. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### 9.4.13 Header (std\_header)

A header bar is a jQuery Mobile toolbar at the top of the page. A header can be positioned at the top of the page or "fixed" to the top of the screen (so it remains in position when the page is scrolled). A header can also be configured to show or hide itself when the user taps on the screen.

A header expects a certain type content and will format that content in a certain way. Specifically, it expects some header text and up to two links or buttons. The header text will be centered and the buttons placed on each side of the header. For example:



If you need to create a header that doesn't follow the default configuration, simply wrap your custom styled markup in any container, such as div. The weblet won't apply the automatic button logic to the wrapped content.

**Note:** The standard shipped layouts will provide a header for you. The header weblet is only required when creating a custom layout, or using a layout that does not provide one.

## Properties - Header (std\_header)

This weblet's properties are:

fullscreenMode id position  
hideIf internal\_id swatch

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **fullscreenMode**

A fullscreen header is a header with a position of "fixed" except that the header overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the header in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **headerPosition** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **position**

Headers can be positioned on the page in a few different ways. By default, the header uses the "inline" positioning mode. In this mode, the header sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the header to the top of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the header will fall back to static, inline position in the page.

## **Default value**

Inline

## **Valid values**

inline or fixed.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

## 9.4.14 HTML List (std\_html\_list)

Lists are used for data display, navigation, result lists, and data entry so jQuery Mobile includes a wide range of list types and formatting examples to cover most common design patterns.

A jQuery Mobile list starts as a simple HTML list (<ul> or <ol>). jQuery Mobile will apply all the necessary styles to transform the list into a mobile-friendly list view with right arrow indicator that fills the full width of the browser window. When you tap on the list item, the framework will trigger a click on the first link inside the list item, issue an AJAX request for the URL in the link, create the new page in the DOM, then kick off a page transition.

The HTML List weblet defines the list and configures various options and list-wide defaults. List items are created by adding one or more **HTML List Item** weblets to the HTML List content.

If lists are embedded in a page with other types of content, an inset list packages the list into a block that sits inside the content area with a bit of margin and rounded corners (theme controlled). Setting the inset property to True on the list weblet applies the inset appearance.



An inset list

with 4 items (one configured as a divider)

Note: all standard, non-inset lists have a -15 pixel margin to negate the 15 pixels of padding on the content area to make lists fill to the edges of the screen. If you add other content above or below a list, the negative margin may make these elements overlap so you'll need to add additional spacing in your custom CSS.

## Properties - HTML List (std\_html\_list)

This weblet's properties are:

class	id	splitIcon
countSwatch	inset	splitSwatch
dividerSwatch	internal_id	swatch
hasSearchFilter	searchFilterSwatch	type
hideIf		

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

## **countSwatch**

Specifies the default jQuery Mobile theme swatch, or color scheme, to apply to any count indicators in the list.

### **Default value**

Default - Uses jQuery Mobile's default swatch (a).

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **dividerSwatch**

Specifies the default jQuery Mobile theme swatch to use for list dividers.

### **Default value**

Default - Uses jQuery Mobile's default swatch (a).

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **hasSearchFilter**

jQuery Mobile provides a very easy way to filter a list with a simple client-side search feature. To make a list filterable, simply set the **hasSearchFilter** property to True. The framework will then append a search box above the list and add the behavior to filter out list items that don't contain the current search string as the user types.

For details of more advanced filtering options, see the [jQuery Mobile documentation](#) on List views.

### **Default value**

False - No search filter is added.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **inset**

If lists are embedded in a page with other types of content, an inset list packages the list into a block that sits inside the content area with a bit of margin and rounded corners (theme controlled). Setting **inset** to True applies the inset appearance.

Note: all standard, non-inset lists have a -15 pixel margin to negate the 15 pixels of padding on the content area to make lists fill to the edges of the screen. If you add other content above or below a list, the negative margin may make these elements overlap so you'll need to add additional spacing in your custom CSS.

## **Default value**

False

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **searchFilterPlaceholder**

Specifies a short hint to display in the search field. The hint is displayed in the field when it is empty, and disappears when the field gets focus or contains data (details vary by browser).

### **Default value**

Blank - jQuery Mobile uses its default of "Filter Items..."

### **Valid values**

Any string value.

## **searchFilterSwatch**

Specifies the default jQuery Mobile theme swatch, or color scheme, to apply to the search field.

### **Default value**

Default - Uses jQuery Mobile's default swatch (a).

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**splitIcon**

Specifies the jQuery Mobile icon to use for the button in a split button list.

**Default value**

Default - a right arrow icon is used

**Valid values**

Any of the values listed in the property dropdown.

## **splitSwatch**

Specifies the default jQuery Mobile theme swatch, or color scheme, to apply to any split buttons in the list.

### **Default value**

Default - Uses jQuery Mobile's default swatch (a).

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**type**

Specifies the type of HTML list to create; an ordered list where each entry is numbered or an unordered list where the entries are not numbered.

**Default Value**

Unordered List

**Valid values**

Ordered List or Unordered List

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### 9.4.15 HTML List Item (std\_html\_li)

The HTML List Item is used to add items to a [9.4.14 HTML List \(std\\_html\\_list\)](#) weblet.

**Tip:** To add items from an RDMLX working list, place an HTML List Item weblet inside an **RDMLX Working List** weblet.

## Properties - HTML List Item (std\_html\_li)

The HTML List Item (std\_html\_li) weblet's properties are:

class    id            role  
filterText internal\_id swatch  
hideIf

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**filterText**

Specifies alternate text to use for this row when filtering. For more details on list filtering, see the [hasSearchFilter](#) property of the HTML List weblet.

**Default value**

Blank - the actual contents of the row are used for filtering.

**Valid values**

Any string value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **role**

List items can be turned into dividers to organize and group the list items. This is done by setting the **role** property to "List divider". A list divider is styled with the bar swatch "a" by default (light grey in the default theme) but you can specify a theme for dividers by setting the **dividerSwatch** property in the parent HTML List weblet.

## **Default value**

[None]

## **Valid Values**

[None] and List divider.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### **9.4.16 Image (std\_image)**

Displays an image. Has the option to load the image only when it comes into view, which helps render the page faster.

## **Properties – Image (std\_image)**

The properties for this weblet are:

caption hideIf relativeImagePath  
height lazyLoad width

## **relativeImagePath**

The path and file name, relative to the images virtual directory, of the image to be displayed.

### **Default value**

Blank – no image is displayed.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **lazyLoad**

If true, the image is not loaded until it comes into view

### **Default value**

True.

### **Valid values**

true(), false() or any valid XPath expression that returns a boolean value.

## **width**

Specifies the width of the input in pixels.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

### **Default value**

Blank - the browser will calculate the width after it loads the image.

### **Valid values**

Any integer value.

## **height**

Specifies the height of the input in pixels.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

### **Default value**

Blank - the browser will calculate the height after it loads the image.

### **Valid values**

Any integer value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**caption**

Specifies alternate text for the user, if he/she for some reason cannot view the image (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

**Default value**

Blank

**Valid values**

Any string value.

### 9.4.17 Loader (std\_loader)

Displays a small loading overlay when jQuery Mobile loads in content via AJAX, or when you want to perform an action that momentarily blocks user interaction.

You use JavaScript to start/stop the loader weblet:

```
Lstd.Weblets.stdLoader.start(my_id); // Displays the loader with an id of "my_id"
```

```
Lstd.Weblets.stdLoader.stop(); // Stops the active loader
```

## Properties – Loader (std\_loader)

The properties for this weblet are:

id            showText

name        text

showIcon   swatch

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**showIcon**

Set to true to show the jQuery Mobile loader icon.

**Default value**

True – icon is shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**showText**

Set to true to show a text message.

**Default value**

False –text message is not shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**text**

The text message to show when the loader is active.

**Default Value**

Auto –Default message "In Progress ..."

**Valid values**

Any string value.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## 9.4.18 Input Box (std\_input)

The Input Box weblet creates an `<input>` element. The `<input>` element is the basic form of data entry in HTML pages. It is used to create every type of input control (including buttons) that HTML supports except multi-line text fields. The jQuery Mobile framework will automatically enhance the default controls for the mobile environment and the theme.

The Input Box weblet supports all HTML 5 input attributes. However, since HTML5 is fairly new and not all the details have been worked out yet, not all web browsers support HTML5 features the same way, or at all. To get the latest information on which browsers support what features, visit the [Wufoo.com](http://Wufoo.com) [HTML 5](#) page.

**Tip:** Although the Input Box is able to create checkboxes and radio buttons, it will usually be easier to use the Checkbox and Radio Button Group weblets as they have been designed to make working with RDMLX easier.

## Properties - Input Box (std\_input)

The Input Box (std\_input) weblet's properties are:

accept	formnovalidate	pattern
addErrorDiv	formtarget	placeholder
alt	height	rdmlxDatatype
autocomplete	hideIf	readonly
autofocus	hideLabel	required
class	id	size
clearButton	label	src
clearButtonText	list	step
corners	max	style
disabled	maxlength	swatch
displayMode	min	tabindex
fieldContainWrapper	mini	title
form	multiple	type
formaction	name	value
formenctype		width
formmethod		

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

Specifies the value of the weblet. For text input types, this is the value displayed in the input. For button, checkbox and radio types, this is the value that will be submitted if the input is selected when the form is submitted.

**Default value**

Blank

**Valid values**

Any string value.

## **accept**

If the input type is File, this property specifies the type of files the server accepts. File types are specified by using a comma separated list of MIME types. Wildcards may be specified so that, for example, you can specify any image file with "image/\*". Go to the [IANA web site](#) for a full list of valid MIME types.

**Note:** Not all browsers support this attribute and browsers may not recognize all MIME types so do not rely on this property to validate files. Always validate them on the server.

**Note:** LANSA servers do not handle inputs of type "file". This property is only useful when submitting to a third party server.

## **Default value**

Blank - all file types are accepted.

## **Valid values**

Comma separated list of MIME types.

## **addErrorDiv**

When set to True, a <div> element will be added just after the weblet to display validation errors. the <div> will be hidden until a validation error occurs and will be hidden again when the error is cleared.

When a validation method has been set the error <div> will reserve some space for itself on the page even when it is hidden to avoid confusing rearrangements of the screen when the error is displayed. If you are using HTML 5 validation on the form but not on this field then you may want to set this property to False to reclaim that space.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **alt**

Specifies alternate text for the user, if he/she for some reason cannot view the image (because of slow connection, an error in the src attribute, or if the user uses a screen reader). This property is only valid when **type**="image".

### **Default value**

Blank

### **Valid values**

Any string value.

## **autocomplete**

Specifies whether or not an input field should have autocomplete enabled. Autocomplete allows the browser to predict the value. When a user starts to type in a field, the browser should display options to fill in the field, based on earlier typed values. The autocomplete property works with the following <input> types: text, search, url, tel, email, password, datepickers, range, and color.

**Note:** This is not an ajax autocomplete. The browser remembers the values previously entered in the field and uses that history to provide the autocomplete functionality. It may be undesirable to have this option turned on on some fields for security reasons (for example, user and password fields) or because it is not helpful to the user (numeric entry fields perhaps).

### **Default value**

Default - uses the browser default. This is usually "on" but may depend on user preferences.

### **Valid values**

on, off or Default. Note that a value of "on" will never override a user preference setting disabling this feature.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**corners**

Specifies if the input field should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **displayMode**

Controls whether the weblet accepts input, is output only, or is hidden. Setting **displayMode** to 'hidden' is equivalent to setting the **type** property to 'hidden'.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

input, output or hidden

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

## **formation**

A URL that specifies where to send the form-data when a form is submitted. Only valid for **type**="submit". Normally, the LANSAs runtime framework takes care of working out the correct URL based on the onClickWrName and onClickWR fields. Setting this property will override this default behaviour.

## **Default Value**

Blank

## **Valid values**

Any valid URL.

## **formenctype**

Specifies the encoding type to use when sending the form. Only valid for **type**="submit". Normally you will not need to set this as the default type is fine for sending most forms to LANSAs servers. If you are sending a form to a non-LANSAs server then specific requirements of that server may require a different encoding type.

### **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of application/x-www-form-urlencoded.

### **Valid values**

application/x-www-form-urlencoded	All characters are encoded before sent (spaces are converted to "+" symbols, and special characters are converted to ASCII HEX values).
multipart/form-data	No characters are encoded. This value is required when you are using forms that have a file upload control.
text/plain	Spaces are converted to "+" symbols, but no special characters are encoded.

## **formmethod**

Specifies which HTTP method to use to send the form data. Only valid for **type="submit"**.

## **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of POST.

## **Valid values**

'GET' or 'POST'.

## **formnovalidate**

Specifies that the form should not be validated prior to submission. Only valid for **type**="submit". Setting this property to True will suppress any validation that has been turned on in the parent <form> element, setting it to False will not turn on validation that has been turned off in the parent <form> element.

For standard shipped layouts the <form> validation is configured with the validationMethod property of the layout weblet.

### **Default value**

False.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **formtarget**

Specifies the frame or window in which to display the response after submitting the form. Only valid for **type="submit"**.

This property is part of the HTML 5 specification and is provided for completeness. Care should be taken with its use. Different mobile browsers have different levels of support for frames and windows and may handle this property differently. It may also cause conflicts with jQuery Mobile's Ajax mechanisms.

### **Default value**

Blank - Use the value specified by the parent <form> element which, unless otherwise specified, will be a default of blank also (display in the current page).

### **Valid values**

\_blank, \_self, \_parent, \_top, framename.

## **height**

Specifies the height of the input in pixels. This property is only valid when **type="image"**.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

## **Default value**

Blank - the browser will calculate the height after it loads the image.

## **Valid values**

Any integer value.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**list**

Specifies the ID of a `<datalist>` element containing a list of values the browser can use as suggestions as the user is typing. This property provides functionality similar to `autocomplete` except that the suggested values are provided by the `<datalist>` element rather than the values the user has previously entered in this field.

**Default value**

Blank

**Valid values**

The ID of a `<datalist>` element in the current document.

## **max**

Specifies the maximum value for an <input> element. Only valid for the following input types: number, range, date, datetime, datetime-local, month, time and week.

**Note:** The exact details of how a browser may use the value may vary. For example, one browser may prevent the user from entering invalid numbers while another may just use the value for field validation.

## **Default value**

Blank - no maximum value is specified.

## **Valid values**

Any value consistent with the <input> type (for example, a number for type=number, a date for type=date).

**maxlength**

Specifies the maximum number of characters allowed in the <input> element. The browser does not distinguish between SBCS and DBCS characters so this property will not limit the data length of the field. If the rdmlxDatatype property is correctly set and form validation turned on, the LANSa framework will validate the data length of the field.

**Default value**

Blank - no maximum length.

**Valid values**

Any integer value.

## **min**

Specifies the minimum value for an <input> element. Only valid for the following input types: number, range, date, datetime, datetime-local, month, time and week.

**Note:** The exact details of how a browser may use the value may vary. For example, one browser may prevent the user from entering invalid numbers while another may just use the value for field validation.

### **Default value**

Blank - no minimum value is specified.

### **Valid values**

Any value consistent with the <input> type (for example, a number for type=number, a date for type=date).

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

**multiple**

Specifies that the user is allowed to enter more than one value in the <input> element. This property is only valid for input types "email" and "file".

**Default value**

False

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **pattern**

Specifies a regular expression that the <input> element's value is checked against. The browser will not allow the form to be submitted if the <input> value does not pass this check. The pattern property works with the following input types: text, search, url, tel, email, and password.

The value used for this property is actually a partial regular expression. A regular expression is normally used to find a bit of text in a larger string. For validation purposes, this pattern must match the whole value, not just a substring. To achieve this, the pattern is modified to anchor it to the beginning and end of the value by adding `^(?:` to the beginning of the pattern and `)$` to the end.

**Tip:** Learn more about JavaScript regular expressions in this [JavaScript tutorial](#).

## **Default value**

Blank

## **Valid values**

A valid JavaScript regular expression.

**placeholder**

Specifies a short hint that describes the expected value of the input field (for example, a sample value or a short description of the expected format). The hint is displayed in the field when it is empty, and disappears when the field gets focus or contains a value (details vary by browser).

**Default value**

Blank

**Valid values**

Any string value.

## **rdmlxDataType**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

## **readonly**

Sets the read-only state of the input. A read-only input field cannot be modified (however, a user can tab to it, highlight it, and copy the text from it).

The readonly attribute can be set to keep a user from changing the value until some other conditions have been met (like selecting a checkbox, etc.). Then, a JavaScript can remove the readonly value, and make the input field editable.

## **Default value**

False - the input is editable.

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## required

Specifies that a value must be entered in this weblet before the form can be submitted. . Note that, at the time of writing, Safari and Internet Explorer do not support this property.

**Note:** The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

**Note:** HTML 5 form validation can be turned off by adding a novalidate attribute to the <form> tag or submit button (this is automatically done by setting validationMethod to 'none' on the standard shipped layouts). In addition, different browsers have differing levels of support for HTML 5 validation and there are many ways for malicious users to bypass client-side validation. So, while client-side validation can improve the user experience, you should never rely on it. Always back it up with server-side validation.

## Default value

False

## Valid values

True, False, or any valid XPath expression that returns a boolean value.

## **size**

Specifies the visible width, in characters, of the <input> element. The size attribute works with the following input types: text, search, tel, url, email, and password.

This only specifies the visible width of the input, not the length of data that can be entered. See the [maxlength](#) property to limit the length of entered data.

The size value is measured in numbers of characters. Each browser has slightly different techniques for determining how to convert this number into an actual number of pixels on screen. For more accurate control of the width of the <input>, use CSS to set the width.

## **Default value**

Blank - the browser uses its own default size, usually around 20.

## **Valid values**

Any integer value.

**src**

Specifies the URL of the image to use as a submit button. This property is only valid when **type**="image".

**Default value**

Blank

**Valid values**

Any valid URL.

## **step**

Specifies the legal number intervals for an <input> element. For example: if step="3", legal numbers could be -3, 0, 3, 6, etc. The step attribute can be used together with the **max** and **min** properties to create a range of legal values. The step property works with the following input types: number, range, date, datetime, datetime-local, month, time and week.

Note: most browsers will default this value to 1. This has the effect of making fractional values invalid. If you are using browser validation and need to allow fractional values, set this to 'any'.

### **Default value**

Blank - no step specified.

### **Valid values**

Any numeric value or 'any'.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. For example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

## **Valid values**

Any valid HTML attribute string.

## type

The type attribute specifies the type of <input> element to display. The full list of types defined in HTML 5 is shown below. Not all types are supported by all browsers. To get the latest information on which browsers support what features, visit the [Wufoo.com HTML 5](#) page.

Note: When using a type of "number" browsers will default the step property to 1 which may prevent the entry of fractional values. Make sure you also set the step property correctly.

<b>Value</b>	<b>Description</b>
button	Defines a clickable button (mostly used with a JavaScript to activate a script)
checkbox	Defines a checkbox
color	Defines a color picker
date	Defines a date control (year, month and day (no time))
datetime	Defines a date and time control (year, month, day, hour, minute, second, and fraction of a second, based on UTC time zone)
datetime-local	Defines a date and time control (year, month, day, hour, minute, second, and fraction of a second (no time zone))
email	Defines a field for an e-mail address
file	Defines a file-select field and a "Browse..." button (for file uploads)
hidden	Defines a hidden input field
image	Defines an image as the submit button
month	Defines a month and year control (no time zone)
number	Defines a field for entering a number
password	Defines a password field (characters are masked)

radio	Defines a radio button
range	Defines a control for entering a number whose exact value is not important (like a slider control)
reset	Defines a reset button (resets all form values to default values)
search	Defines a text field for entering a search string
submit	Defines a submit button
tel	Defines a field for entering a telephone number
text	Default. Defines a single-line text field (default width is 20 characters)
time	Defines a control for entering a time (no time zone)
url	Defines a field for entering a URL
week	Defines a week and year control (no time zone)

### **Default value**

text

### **Valid values**

A value from the table above.

**clearButton**

If true, shows a button to clear the contents of the input field.

**Default value**

False – Clear button not enabled.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **clearButtonText**

The text description for the optional clear button, useful for assistive technologies like screen readers.

### **Default value**

"Clear text".

### **Valid values**

Any string value.

## **width**

Specifies the width of the input in pixels. This property is only valid when **type="image"**.

**Tip:** Always specify both the **height** and **width** properties for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these properties, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

## **Default value**

Blank - the browser will calculate the width after it loads the image.

## **Valid values**

Any integer value.

### 9.4.19 Messages (std\_messages)

The Messages weblet displays messages created with the RDMLX Message command.

The lateral sensor arrays have been reconfigured

## **Properties - Messages (std\_messages)**

This weblet's properties are:

[hideIf](#)

[swatch](#)

**hideIf**

An expression which, if evaluated to be True, will hide the weblet. If False, the weblet will be visible if there are messages to display, hidden otherwise.

**Default value**

False - the weblet will be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

a

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

### **9.4.20 Mobiscroll Date and Time Picker (std\_mobiscroll)**

The Mobiscroll weblet is a customizable date and time picker optimised for mobile devices. Full details can be found on the [Mobiscroll Project site](#).

## Properties - Mobiscroll Date and Time Picker (std\_mobiscroll)

This weblet's properties are:

class	id	stepMinute
dateFormat	label	stepSecond
dateOrder	mode	style
disabled	name	swatch
endYear	pickerTheme	timeFormat
fieldContainWrapper	rows	timeWheels
form	showOnFocus	title
hideIf	startYear	type
hideLabel	stepHour	value

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

## **value**

Specifies the initial value of the weblet. this value must conform to a specific format. If you use an RDMLX date or time field the format will be correct. If you use other fields such as a numeric field with an edit mask, it is your responsibility to ensure the format output by your webroutine is correct. The format to use will depend on the value of the type property:

<b>Type</b>	<b>Format</b>
-------------	---------------

date	yyyy-mm-dd
------	------------

time	HH:ii:ss
------	----------

datetime	yyyy-mm-ddTHH:ii:ss
----------	---------------------

Where yyyy is a 4 digit year value, mm a two digit month value, dd a two digit day value, HH a two digit hour value in 24 hour format, ii a two digit minute value and ss a two digit second value.

## **Default value**

Blank

## **Valid values**

Any valid date, time or datetime value formatted as above.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

## **dateFormat**

Specifies the format for parsed and displayed dates (m - month of year (no leading zero), mm - month of year (two digit), M - month name short, MM - month name long, d - day of month (no leading zero), dd - day of month (two digit), y - year (two digit), yy - year (four digit)).

### **Default value**

Blank – Uses the localised default as defined in the current `std_messages` file (see 3.10 Localization for details).

### **Valid values**

Any string using formatting characters described above.

## **dateOrder**

Specifies the display order and formatting for the month/day/year wheels. (m - month of year (no leading zero), mm - month of year (two digit), M - month name short, MM - month name long, d - day of month (no leading zero), dd - day of month (two digit), D - day of week (short), DD - day of week (long), y - year (two digit), yy - year (four digit). The options also controls if a specific wheel should appear or not, for example, use 'mmyy' to display month and year wheels only, 'mmD ddy' to display both day of week and date on the day wheel.

### **Default value**

Blank – Uses the localised default as defined in the current `std_messages` file (see 3.10 Localization for details).

### **Valid values**

Any string using formatting characters described above.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**endYear**

Specifies the last displayed year on the year wheel.

**Default value**

Blank – uses the current year + 10.

**Valid values**

Any valid 4 digit year.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

## **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

## **Valid values**

Any string value.

**mode**

Specifies the type of picker displayed when the user taps the field. there are two possible types:

scroller – The user swipes the screen to rotate selection wheels.

clickpick – The user taps on +/- buttons to change the displayed values.

**Default value**

scroller

**Valid values**

scroller or clickpick.

## **pickerTheme**

Sets the picker's visual appearance. The supplied themes are: 'jQuery Mobile', 'Android', 'Android ICS', 'Sense-UI' and 'iOS'.

**Tip:** It's possible to create custom themes in css by prefixing any css class used in the scroller markup with the theme name, for example, `.my-theme .dww { / My CSS / }`, and set the **pickerTheme** property to 'my-theme'.

### **Default value**

jQuery Mobile

### **Valid values**

jQuery Mobile, iOS, Android, Android ICS, Sense-UI or any theme defined as described above.

**rows**

Specifies the number of visible rows on the wheel.

**Default value**

3

**Valid values**

Any positive integer. The weblet will attempt to honour whatever number you enter here, however, a value larger than the number that can fit on the users screen will result in alignment issues and an inability to access the set and cancel buttons.

## **showOnFocus**

When true, the picker will automatically display when the field gets the focus. If you set this property to false it is your responsibility to show the picker at a time of your choosing using JavaScript. For example, if the weblet has an id of "MyDatePicker" the following code would display the picker:

```
$("#MyDatePicker").scroller("show");
```

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**startYear**

Specifies the first displayed year on the year wheel.

**Default value**

Blank – uses the current year - 10.

**Valid values**

Any valid 4 digit year.

**stepHour**

Specifies the steps between hours on the timepicker.

**Default value**

1

**Valid values**

Any integer value between 1 and 24.

**stepMinute**

Specifies the steps between minutes on the timepicker.

**Default value**

1

**Valid values**

Any integer value between 1 and 60.

**stepSecond**

Specifies the steps between seconds on the timepicker.

**Default value**

1

**Valid values**

Any integer value between 1 and 60.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **timeFormat**

The format for parsed and displayed times (h - 12 hour format (no leading zero), hh - 12 hour format (leading zero), H - 24 hour format (no leading zero), HH - 24 hour format (leading zero), i - minutes (no leading zero), ii - minutes (leading zero), s - seconds (no leading zero), ss - seconds (leading zero), a - lowercase am/pm, A - uppercase AM/PM).

### **Default value**

Blank – Uses the localised default as defined in the current `std_messages` file (see 3.10 Localization for details).

### **Valid values**

Any string using formatting characters described above.

## **timeWheels**

Specifies the display order and formatting for the hour/minute/second wheels. (h - 12 hour format (no leading zero), hh - 12 hour format (leading zero), H - 24 hour format (no leading zero), HH - 24 hour format (leading zero), i - minutes (no leading zero), ii - minutes (leading zero), s - seconds (no leading zero), ss - seconds (leading zero), a - lowercase am/pm, A - uppercase AM/PM).. The options also controls if a specific wheel should appear or not, for example, use 'HHii' to display hours (24 hour format) and minutes wheels only, 'hhiissa' to display hours (12 hour format), minutes, seconds and am/pm wheels.

### **Default value**

Blank – Uses the localised default as defined in the current `std_messages` file (see 3.10 Localization for details).

### **Valid values**

Any string using formatting characters described above.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

## **Valid values**

Any valid HTML attribute string.

**type**

Specifies the data type being displayed in the weblet.

**Default value**

date

**Valid values**

date, time or datetime.

## 9.4.21 Navigation Bar (std\_navbar)

jQuery Mobile has a very basic navbar widget that is useful for providing up to 5 buttons with optional icons in a bar, typically within a header or footer. There is also a persistent navbar variation that works more like a tab bar that stays fixed as you navigate across pages.

A Navigation bar is created using an unordered HTML list (<ul>). To add a button to the Navigation bar, add a list item (use the HTML List Item weblet or insert the <li></li> directly into the source) and then place an Anchor into the list item. You can add any number of buttons to a Navigation bar. If you add more than 5 the Navigation Bar will simply wrap to multiple lines.



**Tip:** to make a button appear as selected, add the class "ui-btn-active" to the corresponding icon.

## Properties - Navigation Bar (std\_navbar)

This weblet's properties are:

<a href="#">id</a>	<a href="#">iconPosition</a>
<a href="#">class</a>	<a href="#">swatch</a>
<a href="#">hideIf</a>	<a href="#">internal_id</a>

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

To make the Navigation Bar persistent, add the value "ui-state-persist" to the class.

## **Default value**

Blank

## **Valid values**

A string containing one or more space separated CSS class names.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **iconPosition**

Specifies the position of the button icon relative to the button text in any buttons in the Navbar.

### **Default value**

Blank - uses the jQuery Mobile default position of 'Left'.

### **Valid values**

Left - position the icon to the left of the button text.

Right - position the icon to the right of the button text.

Top - position the icon above the button text.

Bottom - position the icon below the button text.

No Text - draw the button with no text (icon only).

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

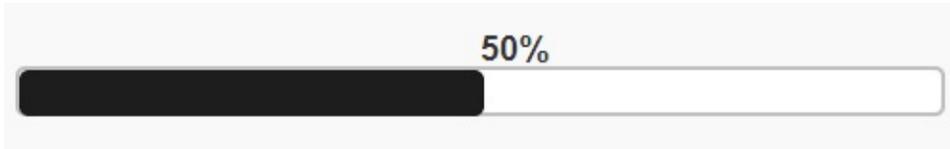
Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### 9.4.22 Progress bar (std\_progressbar)

Display status of a determinate process. It can also be used to display a value as a percentage of its maximum value.



Note : For indeterminate progress, use the Loader weblet (std\_loader) that provides the standard jQuery Mobile icon (with an optional text message).

## Properties – Progress bar

The properties for this weblet are:

caption    max    swatch  
delayClose name    transitory  
hideIf    overlay value

**name**

The name of the weblet. If the weblet represents a field from your WAM, this should be the name of the field. If the weblet does not represent a field, you may wish to use your own name if using JavaScript or XSL that references the weblet. If nominating a name is not a consideration, the default name should be used, as determined by LANSA.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name, in single quotes. This can be a nominal choice, or the name of a field, the value of which is set by the weblet.

**value**

The value to set the weblet to. The ratio of this value to the maximum value is used to represent the current progress.

**Default value**

Zero: 0% of progress

**Valid values**

0 to maximum value.

**max**

The value that represents 100% of progress. This, together with the current value are used to determine the progress shown by the progress bar.

**Default value**

100

**Valid values**

A numeric value greater than zero.

**caption**

Caption to show on progress bar once it reaches its maximum value.

**Default value**

Auto: "Complete"

**Valid values**

Single-quoted text.

**transitory**

Show progress bar only while its value is greater than zero until it reaches its maximum value.

**Default value**

False()

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **overlay**

For transitory progress bars: Overlay the progress bar at the center of the viewport. This prevents users from interacting with the page while the progress bar is displayed.

### **Default value**

False()

### **Valid values**

Any valid XPath expression that returns a Boolean value.

## **delayClose**

For transitory progress bars: Delay time (in milliseconds) to keep the progress bar before closing it.

### **Default value**

500

### **Valid values**

An integer value

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

### 9.4.23 Radio Button Group (std\_radbuttons)

The radio button group weblet creates a set of radio buttons linked to an RDMLX field. The number of buttons and button values can be established from a working list or a static set of values defined via the **items** property of the weblet.



Choose a pet:

- Cat
- Dog
- Hamster
- Lizard

## Properties - Radio Button Group (std\_radbuttons)

This weblet's properties are:

id	fieldContainWrapper	label
name	form	mini
value	hideIf	orientation
disabled	hideLabel	rdmlxDataType
displayMode	items	selectorValueEq
		swatch

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

Specifies the value of the weblet. The weblet will compare this value with the code value of each item to determine which radio button to set as initially selected.

**Default value**

Blank

**Valid values**

Any string value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**displayMode**

Controls whether the weblet accepts input or is output only.

**Default value**

Blank (equivalent to 'input').

**Valid values**

input or output.

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label. It is currently not possible to hide the label for this weblet so that it is still accessible to assistive technologies (as the **hideLabel** property will do on other weblets) so setting this property to true is the same as setting the **label** property to a blank value. This property exists for consistency with other weblets and so that, should future browsers make it possible to hide the label accessibly, that will automatically happen without you needing to change anything.

### **Default value**

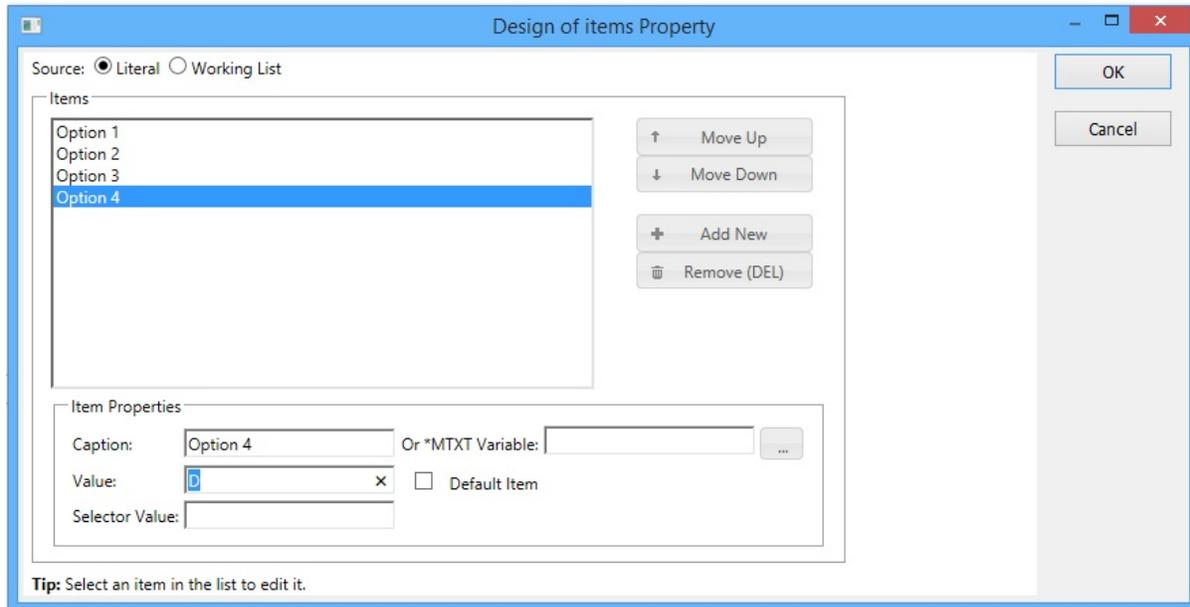
False - the label is not hidden.

### **Valid values**

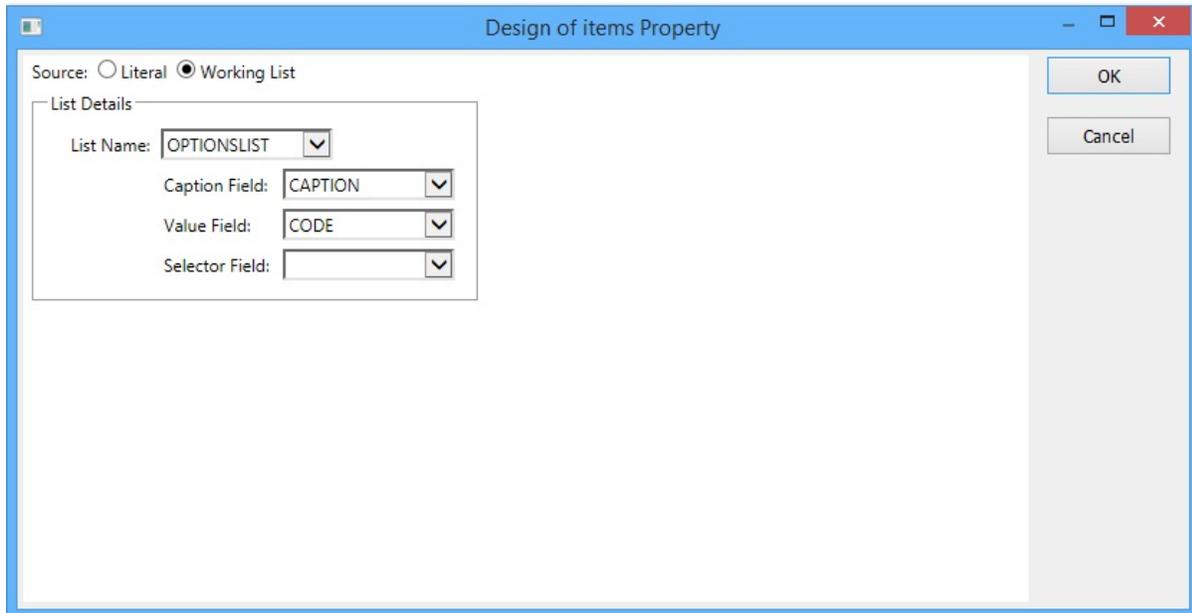
True, False, or any valid XPath expression that returns a boolean value.

## items

An XML nodeset specifying the items to appear in the weblet. This can only be set by the designer. To invoke the designer use the ellipse button in the property sheet. The property designer can be used to specify a hard coded set of items or the name of a working list to get the items from.



This shows a list configured with 4 items. Check the Default Item check box for the item which is to be selected if no value is preselected. The Selector value can be used to filter the list down to a smaller set of displayed values at runtime.



This shows the items property editor configured to use a working list. The Selector field can be used to filter the list down to a smaller set of displayed values at runtime.

### **Default value**

document("")/\*/lxml:data/lxml:radiobutton (this indicates no items have been defined for this weblet.)

### **Valid values**

Not Applicable. (This value is system maintained.) To invoke the designer use the ellipse button in the property sheet.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

**orientation**

Specifies how the radio items should be arranged relative to each other.

**Default value**

vertical

**Valid values**

horizontal or vertical.

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **rdmlxDataType**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

**selectorValueEq**

This value is used in order to limit, to a subset, the list items shown in the weblet. A selector value or selector field must have been specified in the items property editor.

**Default value**

Blank.

**Valid values**

Any string value.

#### **9.4.24 RDMLX Working List (std\_repeater)**

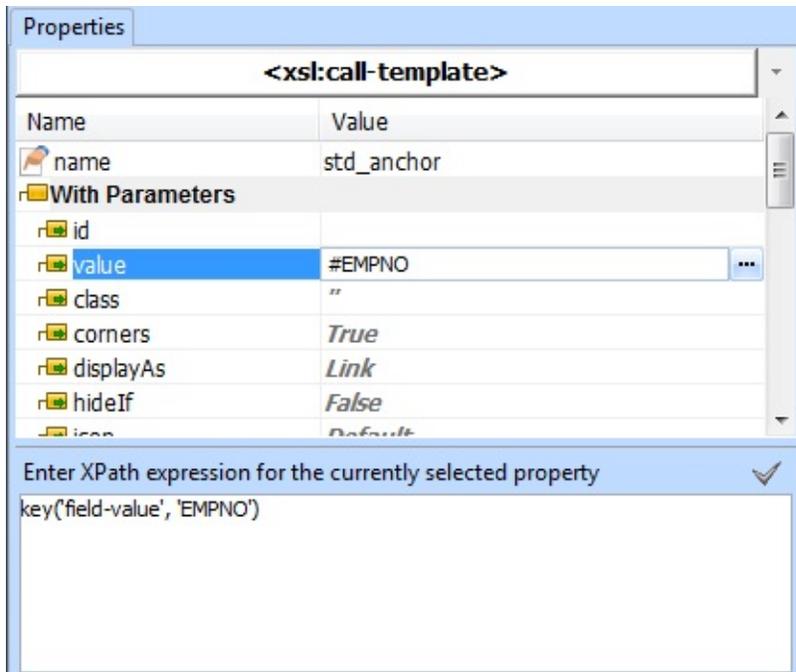
The RDMLX Working List weblet provides a flexible mechanism to process working lists. It does not generate any output directly. It repeats whatever content you place in its content area once for each row in the working list.

The most common approach to handling repeating content in modern semantic HTML pages is to place it in an HTML list (<ul> or <ol>) and wrap each "row" of repeating content with list item (<li>) tags. This is the approach taken by jQuery Mobile lists. The RDMLX working list weblet allows you to create any repeating content you like such as table rows, divs, etc.

To create a jQuery Mobile list, start by dropping an HTML List weblit onto your design. Then drop a RDMLX Working List weblet into the content area of the HTML List. Set the listname property to the name of your working list. Next, drop an HTML List Item weblet into the working list content area. Finally, place the content you want repeated for each row into the list item content area.

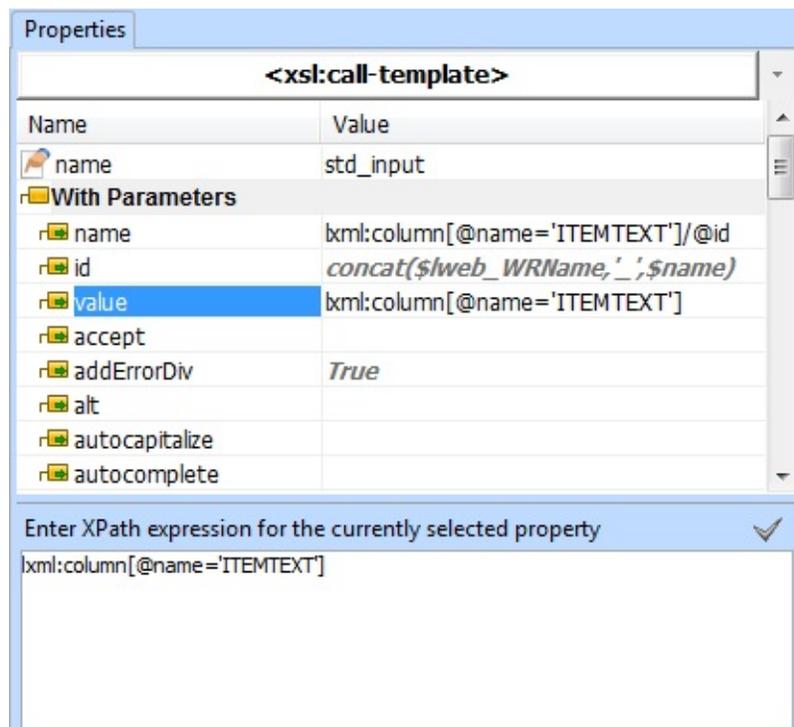
#### **Referencing column values**

The data from your webroutine is output as an XML document and passed to the webroutine design for converting into the final HTML output. In order to access the webroutine output, the design needs an XPath expression to tell it where to find the required data in the XML. When you need to access data from a field, you don't normally need to worry about this. You just enter the field name in the property editor (e.g #EMPNO) and the LANSAs Editor automatically converts that into the necessary XPath (for example, key('field-value', 'EMPNO')).



Property Tab showing a field name in the property value and the corresponding XPath expression in the XPath entry box below.

It is not possible to use this shorthand technique when referencing columns in a list. Instead, you need to reference the column with `lxml:column[@name='COLNAME']`



Property tab showing a column value being referenced.

XPath expressions such as this must be entered into XPath entry area at the bottom of the properties tab. Attempting to enter the expression in the property directly will result in incorrect behavior or an error as the editor will attempt to treat it as a string.

**Note:** All field and column name references in XPath expressions must be uppercase. All references to repository fields must use the object name for the field.

## **Properties - RDMLX Working List (std\_repeater)**

This weblet's properties are:

[internal\\_id](#)

[isOutputOnly](#)

[listname](#)

## **isOutputOnly**

Indicates that the list specified by the listname property is only being used by this weblet for output. When a list is being used for input, the LANSAs framework needs to create some hidden fields and do some extra processing to make sure the list data is sent to the server correctly. If the list is being used by this weblet for output purposes only, you can slightly improve performance and reduce the risk of conflicts with other weblets using the same list if you set this property to True.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**listname**

The name of the RDMLX working list to iterate over. The weblet content will be reproduced for each row in the list. If no list is specified then a single row of cells will be created.

**Default value**

Blank

**Valid values**

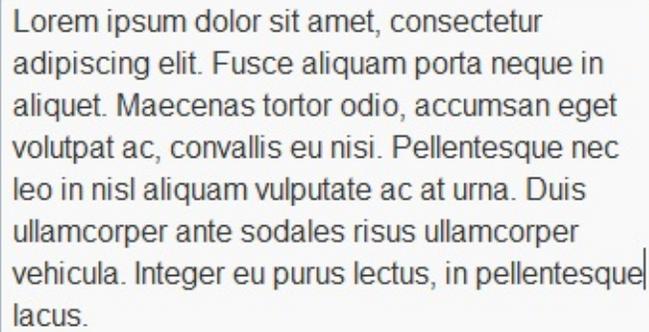
The name of a working list output by the current webroutine. A list of available working lists (as defined in the WAM) can be selected from by clicking the corresponding dropdown button in the property sheet.

**internal\_id**

A unique ID used by the WAM Editor and the weblet to connect the weblet to custom content contained within the webroutine design. This property is automatically configured by the WAM Editor and should not be modified manually.

### 9.4.25 HTML Textarea (std\_textarea)

The HTML Textarea creates a <textarea> element which can be used for the display and input of long text values spanning multiple lines.

A screenshot of a text area containing Lorem Ipsum text. The text is displayed in a light gray box with a blue border and rounded corners. The text is as follows:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce aliquam porta neque in aliquet. Maecenas tortor odio, accumsan eget volutpat ac, convallis eu nisi. Pellentesque nec leo in nisl aliquam vulputate ac at urna. Duis ullamcorper ante sodales risus ullamcorper vehicula. Integer eu purus lectus, in pellentesque lacus.

## Properties - HTML Textarea (std\_textarea)

The HTML Textarea (std\_textarea) weblet's properties are:

<a href="#">addErrorDiv</a>	<a href="#">hideIf</a>	<a href="#">readonly</a>
<a href="#">autofocus</a>	<a href="#">hideLabel</a>	<a href="#">required</a>
<a href="#">class</a>	<a href="#">id</a>	<a href="#">rows</a>
<a href="#">cols</a>	<a href="#">label</a>	<a href="#">style</a>
<a href="#">corners</a>	<a href="#">maxlength</a>	<a href="#">swatch</a>
<a href="#">disabled</a>	<a href="#">mini</a>	<a href="#">tabindex</a>
<a href="#">displayMode</a>	<a href="#">name</a>	<a href="#">title</a>
<a href="#">fieldContainWrapper</a>	<a href="#">placeholder</a>	<a href="#">value</a>
<a href="#">form</a>	<a href="#">rdmlxDataType</a>	<a href="#">wrap</a>

## id

A unique ID for the weblet. This property is not required but, if supplied, allows the weblet to be directly referenced by CSS or JavaScript. The value must be unique within the page.

**Note:** jQuery Mobile loads pages using Ajax and inserts them into the current page, optionally performing an animation as it does. This means that the content for two webrouines may both exist in a page for a brief period of time. If both webrouines contain a weblet with the same ID and you have any custom CSS or JavaScript that references the weblet during this period of time you may get unexpected results. For this reason, you should aim to make an ID globally unique. Weblets that are automatically generated to represent fields do this by concatenating the name of the webrouine with the name of the field.

### Default value

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

### Valid values

The rules for valid characters and formatting in an ID attribute vary depending on the version of HTML you want to support. For specific details, look at the appropriate specifications: <http://www.w3.org/>

To ensure compatibility with all versions of HTML, CSS and JavaScript libraries such as jQuery, you should stick to the following rules:

- Must begin with a letter A-Z or a-z
- Can be followed by: letters (A-Za-z), digits (0-9), hyphens ("-"), and underscores ("\_")

## **name**

The name of the field that will receive the value of this weblet when it is submitted to a webroutine. When a weblet is generated, or dropped on a field from the current webroutine's WEB\_MAP, this property will be set to the name of that field. If the field that you want the value submitted to has a different name, you should change this property to that name.

**Note:** In the XHTML Technology Service, the name property is often used as a unique ID. This is not the case for the jQuery mobile technology service. The id property should be used for that. If the weblet value is not to be submitted to the server, this property can be left blank.

## **Default value**

Blank. Weblets that are automatically generated to represent fields will have the value automatically set.

## **Valid values**

The name of a field in a target webroutine.

**value**

Specifies the text displayed in the textarea.

**Default value**

Blank

**Valid values**

Any string value.

## **addErrorDiv**

When set to True, a <div> element will be added just after the weblet to display validation errors. the <div> will be hidden until a validation error occurs and will be hidden again when the error is cleared.

When a validation method has been set the error <div> will reserve some space for itself on the page even when it is hidden to avoid confusing rearrangements of the screen when the error is displayed. If you are using HTML 5 validation on the form but not on this field then you may want to set this property to False to reclaim that space.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **autofocus**

Specifies that the weblet should automatically get the focus when the page loads. There must be only one field on a page with this property set to true. Setting autofocus to true on more than one field may produce inconsistent results on different browsers.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**class**

The CSS class, or classes, to assign to the weblet. A CSS class allows you to specify a set of CSS styles, defined in an external stylesheet, to apply to a weblet, or elements within a weblet. For complex weblets made from multiple HTML elements, the class is applied to the outermost element of the weblet.

**Default value**

Blank

**Valid values**

A string containing one or more space separated CSS class names.

**corners**

Specifies if the input field should have rounded corners.

**Default value**

True

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **cols**

Specifies the visible width, in characters, of the `<textarea>` element. This only specifies the visible width of the input, not the length of data that can be entered. See the [maxlength](#) property to limit the length of entered data.

The `cols` value is measured in numbers of characters. Each browser has slightly different techniques for determining how to convert this number into an actual number of pixels on screen. For more accurate control of the width of the `<textarea>`, use CSS to set the width.

### **Default value**

Blank - the browser uses its own default size, usually around 20.

### **Valid values**

Any integer value.

## **disabled**

Specifies if the weblet should be disabled. A disabled weblet is unusable and un-clickable. Note that the value of a disabled weblet will not be submitted with the form.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

**displayMode**

Controls whether the weblet accepts input, is output only or is hidden.

**Default value**

Blank (equivalent to 'input').

**Valid values**

input, output or hidden

## **fieldContainWrapper**

jQuery Mobile will handle all the complexities of laying out labels and fields vertically on small screens and so they all line up on wide screens. To do this it needs each label/field pair to be wrapped in a <div> tag with specific attributes. Setting fieldContainWrapper to true and using the label property to set the label text will take care of all of this for you.

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **form**

A space-separated list of form IDs that specifies the form(s) this weblet belongs to. When a form is submitted by clicking a submit button, all the fields that belong to the form are sent to the server. By default, all the fields that are inside the <form> tag belong to the form. This property allows you place a field in other parts of the document, outside of the <form> tag, or inside other <form> tags and still have its value submitted with the form.

The standard LANSA layouts contain a single <form> tag that wraps the entire page so it is usually not necessary to use this property on these layouts.

### **Default value**

Blank

### **Valid values**

Space separated list of form IDs.

**hideIf**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False - the weblet will always be shown.

**Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **hideLabel**

Hides the label accessibly. This means that the label is not visible but it is still available to assistive technologies like screen readers.

### **Default value**

False - the label is not hidden.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **label**

Specifies the label text to use for the weblet. The weblet will create a <label> tag with this value and make sure it is correctly attached to the input field. For the sake of accessibility, it is recommended that you provide a meaningful label for all weblets even if you do not intend to display the label. Use the `hideLabel` property to hide the label while keeping it available for assistive technologies.

### **Default Value**

Blank - Automatically generated fields will have a value from the repository definition.

### **Valid values**

Any string value.

**maxlength**

Specifies the maximum number of characters allowed in the textarea. The browser does not distinguish between SBCS and DBCS characters so this property will not limit the data length of the field. If the `rdmlxDatatype` property is correctly set and form validation turned on, the LANSAs framework will validate the data length of the field.

**Default value**

Blank - no maximum length.

**Valid values**

Any integer value.

**mini**

If set to true, this will display a more compact version of the weblet that uses less vertical height. This can be useful in toolbars and other places where space is tight.

**Default value**

False - the standard size is used.

**Valid Values**

True, False, or any valid XPath expression that returns a boolean value.

## **placeholder**

Specifies a short hint that describes the expected value of the input field (for example, a sample value or a short description of the expected format). The hint is displayed in the field when it is empty, and disappears when the field gets focus or contains a value (details vary by browser).

## **Default value**

Blank

## **Valid values**

Any string value.

## **rdmlxDatatype**

Specifies the RDMLX data type of the field associated with the weblet. This helps some weblets perform data validation. This property is normally set automatically when you generate or drop a field onto a design. You may need to set it yourself if you drop the weblet onto a design and then later associate it with a field.

### **Default value**

Blank unless automatically set by the WAM editor.

### **Valid values**

A | delimited string starting with the data type followed by extra parameters as required by the data type:

integer|<max length>

float|<max length>

packed|<total digits>|<fraction digits>|<decimal separator>

signed|<total digits>|<fraction digits>|<decimal separator>

dec|<total digits>|<fraction digits>|<decimal separator>

alpha|<keyboard shift>|<max length>

char|<keyboard shift>|<max length>

varchar|<keyboard shift>|<max length>

nchar|<keyboard shift>|<max length>

nvarchar|<keyboard shift>|<max length>

## **readonly**

Sets the read-only state of the textarea. A read-only field cannot be modified (however, a user can tab to it, highlight it, and copy the text from it).

The readonly attribute can be set to keep a user from changing the value until some other conditions have been met (like selecting a checkbox, etc.). Then, a JavaScript can remove the readonly value, and make the textarea field editable.

## **Default value**

False - the textarea is editable.

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **required**

Specifies that a value must be entered in this weblet before the form can be submitted. . Note that, at the time of writing, Safari and Internet Explorer do not support this property.

**Note:** HTML 5 form validation can be turned off by adding a novalidate attribute to the <form> tag or submit button (this is automatically done by setting validationMethod to 'none' on the standard shipped layouts). In addition, different browsers have differing levels of support for HTML 5 validation and there are many ways for malicious users to bypass client-side validation. So, while client-side validation can improve the user experience, you should never rely on it. Always back it up with server-side validation.

## **Default value**

False

## **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **rows**

Specifies the visible height, in lines, of the `<textarea>` element. This only specifies the visible height of the textarea, not the length of data that can be entered. See the [maxlength](#) property to limit the length of entered data. The textarea will automatically add a scrollbar if necessary.

The **rows** value is measured in numbers of lines. The exact height will vary depending on the font settings being used. For more accurate control of the height of the `<textarea>`, use CSS to set the height.

### **Default value**

Blank - the browser uses its own default size, usually 2.

### **Valid values**

Any integer value.

## **style**

Specifies a CSS style string to apply to the weblet. This property allows you to set CSS style properties for this weblet that will override any values defined in the layout stylesheets.

### **Default value**

Blank

### **Valid values**

Any valid CSS properties and values, separated by semi-colons

## **swatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the weblet.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **tabindex**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this may not be supported in some older browsers.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid integer value.

## **title**

Specifies extra advisory information about the weblet. This is usually extra non-essential information to help a user understand the purpose of the weblet. Different browsers may handle it in different ways. for example, most desktop browsers will display it as a tooltip when the mouse hovers over the weblet. Assistive technologies like screen readers will read it to the user. At the time of this writing, mobile device browsers will ignore it.

## **Default value**

Blank

## **Valid values**

Any valid HTML attribute string.

## **wrap**

Specifies how the text in a text area is to be wrapped when submitted in a form. The property takes two values; "soft" or "hard".

When set to "soft" no extra wrapping is done. Only line breaks inserted by the user are sent to the server.

When set to "hard" the text will be "word wrapped" (that is, wrapped at the nearest word boundary before the maximum line length) by using the value of the [cols](#) property as the maximum line length.

<p><b>Note:</b> this property affects how the text is formatted for submission to the server. It has no effect on how the text is displayed in the browser.</p>
---

### **Default value**

soft - no wrapping is done

### **Valid values**

soft or hard.

## 9.5 Layout Weblets

[Basic Layout \(std\\_layout\\_v2\)](#) A simple one-column layout with an optional header and footer.

[Flexible Layout \(std\\_flex\\_layout\)](#) A layout weblet with two content areas: A main content area and a secondary or sidebar content area. Automatically adjusts to fit the content area on small (phone) screens.

## **9.5.1 Basic Layout (std\_layout\_v2)**

The Basic Layout weblet is a simple one-column layout with an optional header and footer.

## Properties - Basic Layout (std\_layout\_v2)

This weblet's properties are:

<a href="#">addBackButton</a>	<a href="#">footerSwatch</a>	<a href="#">showFooter</a>
<a href="#">backButtonSwatch</a>	<a href="#">headerFullscreenMode</a>	<a href="#">showHeader</a>
<a href="#">backButtonText</a>	<a href="#">headerPosition</a>	<a href="#">showMessages</a>
<a href="#">contentSwatch (deprecated)</a>	<a href="#">headerSwatch</a>	<a href="#">validationErrorDisplay</a>
<a href="#">footerFullscreenMode</a>	<a href="#">pageSwatch</a>	<a href="#">validationTime</a>
<a href="#">footerPosition</a>	<a href="#">persistentFooterId</a>	<a href="#">windowTitle</a>

## **addBackButton**

jQuery Mobile has a feature to automatically create and append "back" buttons to any header. The framework only adds a back button to pages that have been loaded via ajax. Any page loaded completely by the browser is treated as the "home" page and does not have a back button.

This is primarily useful in chromeless installed applications, such as those running in a native app webview or web applications where you have control of the application entry point. When people may enter your application at different points from search engine or from bookmarks you should avoid back buttons and create buttons that explicitly state where tapping them will go.

### **Default value**

False - no back button is added to headers.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **backButtonSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the back button.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **backButtonText**

Specifies the text to display in any added back button.

### **Default value**

Blank - jQuery Mobile will use its default of "Back".

### **Valid values**

Any string value.

## **contentSwatch (deprecated)**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the content area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **footerFullscreenMode**

A fullscreen footer is a footer with a position of "fixed" except that the footer overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the footer in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **footerPosition** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **footerPosition**

Footers can be positioned on the page in a few different ways. By default, the footer uses the "inline" positioning mode. In this mode, the footer sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the footer to the bottom of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the footer will fall back to static, inline position in the page.

### **Default value**

Inline

### **Valid values**

inline or fixed.

## **footerSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the footer area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **headerFullscreenMode**

A fullscreen header is a header with a position of "fixed" except that the header overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the header in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **headerPosition** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **headerPosition**

Headers can be positioned on the page in a few different ways. By default, the header uses the "inline" positioning mode. In this mode, the header sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the header to the top of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the header will fall back to static, inline position in the page.

### **Default value**

Inline

### **Valid values**

inline or fixed.

## **headerSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the header area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **pageSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the whole page (including header, content and footer).

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **persistentFooterId**

In situations where the footer is a global navigation element, you may want it to appear fixed so it doesn't scroll out of view. It's also possible to make a fixed footer persistent so it appears to not move between page transitions. This can be accomplished by using the persistent footer feature included in jQuery Mobile.

To make a footer persistent between transitions, assign an id value to the **persistentFooterId** of all relevant pages and use the same id value for each. For example, by setting **persistentFooterId** to "myfooter" to the current page and the target page, the framework will keep the footer anchors in the same spot during the page animation. This effect will only work correctly if the footers are set to position="fixed" so they are in view during the transition.

### **Default value**

Blank - the footer is not persistent.

### **Valid values**

Any string value.

## **showFooter**

If True, the layout will add a jQuery mobile footer bar to the bottom of the layout. See the "Toolbars" section of the [jQuery Mobile](#) documentation for more details.

The footer bar is a customizable content area. To customize it, right-click on the footer and select a option from the "Content area for content.footer" menu.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **showHeader**

If True, the layout will add a jQuery mobile header bar to the bottom of the layout. See the "Toolbars" section of the [jQuery Mobile](#) documentation for more details.

The header bar is a customizable content area. To customize it, right-click on the header and select a option from the "Content area for content.header" menu.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **showMessages**

If True, the Messages weblet will be placed at the top of the content area to display any messages from the Webroutine.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **validationErrorDisplay**

Specifies how validation errors are displayed. See [Field Validation](#) for more details.

### **Default value**

Blank - use the browsers' default mechanism

### **Valid values**

'Browser Default' or 'Error Div'

## **validationTime**

Specifies when validation errors are displayed. See [Field Validation](#) for more details.

## **Default Value**

After Submit - validation is done when the user presses the submit button.

## **Valid Values**

Immediately, After Focus or After Submit.

## **windowTitle**

Specifies a title for the page. The value of this property is used to set the <title> tag of the page.

### **Default value**

`$lweb_context/lxml:webroutine-title` - this is an XPath expression that references the Webroutine description.

### **Valid values**

Any string value.

## **9.5.2 Flexible Layout (`std_flex_layout`)**

Flexible Layout is a layout weblet with two content areas: A main content area and a secondary or sidebar content area. The layout automatically adjusts the position of the secondary content area on small (phone) screens.

## Properties - Flexible Layout (std\_flex\_layout)

The Flexible Layout (std\_flex\_layout) weblet's properties are:

<a href="#">addBackButton</a>	<a href="#">footerSwatch</a>	<a href="#">showFooter</a>
<a href="#">backButtonSwatch</a>	<a href="#">headerFullscreenMode</a>	<a href="#">showHeader</a>
<a href="#">backButtonText</a>	<a href="#">headerPosition</a>	<a href="#">showMessages</a>
<a href="#">contentSwatch</a>	<a href="#">headerSwatch</a>	<a href="#">sidebarPositionSmallScreen</a>
<a href="#">(deprecated)</a>	<a href="#">pageSwatch</a>	<a href="#">validationErrorDisplay</a>
<a href="#">footerFullscreenMode</a>	<a href="#">persistentFooterId</a>	<a href="#">validationTime</a>
<a href="#">footerPosition</a>		<a href="#">windowTitle</a>

## **addBackButton**

jQuery Mobile has a feature to automatically create and append "back" buttons to any header. The framework only adds a back button to pages that have been loaded via ajax. Any page loaded completely by the browser is treated as the "home" page and does not have a back button.

This is primarily useful in chromeless installed applications, such as those running in a native app webview or web applications where you have control of the application entry point. When people may enter your application at different points from search engine or from bookmarks you should avoid back buttons and create buttons that explicitly state where tapping them will go.

### **Default value**

False - no back button is added to headers.

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **backButtonSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the back button.

### **Default value**

Default - Uses the weblet's default swatch or inherits the swatch from the container.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **backButtonText**

Specifies the text to display in any added back button.

### **Default value**

Blank - jQuery Mobile will use its default of "Back".

### **Valid values**

Any string value.

## **contentSwatch (deprecated)**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the content area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **footerFullscreenMode**

A fullscreen footer is a footer with a position of "fixed" except that the footer overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the footer in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **footerPosition** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **footerPosition**

Footers can be positioned on the page in a few different ways. By default, the footer uses the "inline" positioning mode. In this mode, the footer sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the footer to the bottom of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the footer will fall back to static, inline position in the page.

### **Default value**

Inline

### **Valid values**

inline or fixed.

## **footerSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the footer area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped theme only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **headerFullscreenMode**

A fullscreen header is a header with a position of "fixed" except that the header overlays the page content, rather than reserving a place in the document. This is useful for immersive apps like photo or video viewers where you want the content to fill the whole screen and toolbars can be hidden or summoned to appear by tapping the screen. Keep in mind that the header in this mode will sit over page content so this is best used for specific situations.

This property is ignored unless **headerPosition** is set to "fixed".

### **Default value**

False

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **headerPosition**

Headers can be positioned on the page in a few different ways. By default, the header uses the "inline" positioning mode. In this mode, the header sits in the natural document flow (the default HTML behavior), which ensures that it is visible on all devices, regardless of JavaScript and CSS positioning support.

A "fixed" positioning mode fixes the header to the top of the viewport on browsers that support CSS fixed positioning (which includes most desktop browsers, iOS5+, Android 2.2+, BlackBerry 6, and others). In browsers that don't support fixed positioning, the header will fall back to static, inline position in the page.

### **Default value**

Inline

### **Valid values**

inline or fixed.

## **headerSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the header area of the layout.

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **pageSwatch**

Specifies the jQuery Mobile theme swatch, or color scheme, to apply to the whole page (including header, content and footer).

### **Default value**

Default - Uses the jQuery mobile's default swatch.

### **Valid values**

Any single letter from a-z. The shipped them only contains swatches a-b. See the [jQuery Mobile documentation](#) on themes for details of how to create your own themes.

## **persistentFooterId**

In situations where the footer is a global navigation element, you may want it to appear fixed so it doesn't scroll out of view. It's also possible to make a fixed footer persistent so it appears to not move between page transitions. This can be accomplished by using the persistent footer feature included in jQuery Mobile.

To make a footer persistent between transitions, assign an id value to the **persistentFooterId** of all relevant pages and use the same id value for each. For example, by setting **persistentFooterId** to "myfooter" to the current page and the target page, the framework will keep the footer anchors in the same spot during the page animation. This effect will only work correctly if the footers are set to position="fixed" so they are in view during the transition.

### **Default value**

Blank - the footer is not persistent.

### **Valid values**

Any string value.

## **showFooter**

If True, the layout will add a jQuery mobile footer bar to the bottom of the layout. See the "Toolbars" section of the [jQuery Mobile](#) documentation for more details.

The footer bar is a customizable content area. To customize it, right-click on the footer and select a option from the "Content area for content.footer" menu.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **showHeader**

If True, the layout will add a jQuery mobile header bar to the bottom of the layout. See the "Toolbars" section of the [jQuery Mobile](#) documentation for more details.

The header bar is a customizable content area. To customize it, right-click on the header and select a option from the "Content area for content.header" menu.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **showMessages**

If True, the Messages weblet will be placed at the top of the content area to display any messages from the Webroutine.

### **Default value**

True

### **Valid values**

True, False, or any valid XPath expression that returns a boolean value.

## **sidebarPositionSmallScreen**

Tells the layout how to position the sidebar content on screens that are too small for it to be displayed at the side of the main content (for example, phone screens).

### **Default value**

Top

### **Valid values**

top, bottom or hidden.

## **validationErrorDisplay**

Specifies how validation errors are displayed. See [Field Validation](#) for more details.

### **Default value**

Blank - use the browsers' default mechanism

### **Valid values**

'Browser Default' or 'Error Div'

## **validationTime**

Specifies when validation errors are displayed. See [Field Validation](#) for more details.

## **Default Value**

After Submit - validation is done when the user presses the submit button.

## **Valid Values**

Immediately, After Focus or After Submit.

## **windowTitle**

Specifies a title for the page. The value of this property is used to set the <title> tag of the page.

### **Default value**

`$lweb_context/lxml:webroutine-title` - this is an XPath expression that references the Webroutine description.

### **Valid values**

Any string value.

## 9.6 Utility Weblets

<b>Weblet name</b>	<b>Description</b>
Standard hidden fields (std_hidden)	For LANSA Product Centre's internal use only. Refer to <a href="#">Hidden</a> for information.
Standard locale (std_locale)	For LANSA Product Centre's internal use only. Refer to <a href="#">Local</a> for information.
Standard script (std_script)	For LANSA Product Centre's internal use only. Refer to <a href="#">JavaScript and the Script Weblet</a> for information.
End of Page Scripts (scriptAtEnd)	For LANSA Product Centre's internal use only. Refer to <a href="#">JavaScript and the Script Weblet</a> for information.
Standard variables (std_variables)	For LANSA Product Centre's internal use only. Refer to <a href="#">Variables</a> for information.
std_keys	A set of xsl:key elements used by field references in XSL. Refer to <a href="#">Keys</a> for information.
std_types	For LANSA Product Centre's internal use only. Refer to <a href="#">Types</a> for information.
std_util	For LANSA Product Centre's internal use only.
CSS Styles (std_style_v2)	Default style that is used for all generated WEBROUTINE pages.

## WAM Tutorials

If you haven't read the other sections of the *Web Application Modules Guide*, you will find an outline of the things you need to understand before you can obtain any benefit from the following exercises in the topic [What is a WAM?](#)

Before you start the exercises, make sure you have the material you require as listed in [Before You Begin](#).

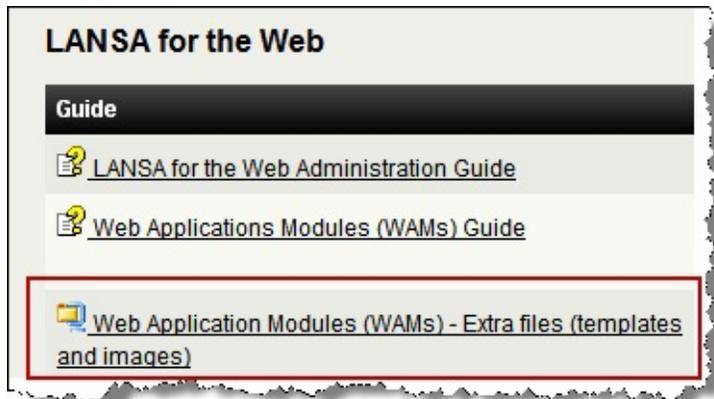
The exercises provided are:

- [WAM005 - Create Your First WAM](#)
- [WAM010 - Using WEB\\_MAPs](#)
- [WAM015 - Working Lists](#)
- [WAM020 - WAM Navigation](#)
- [WAM025 - Using the Layout Wizard](#)
- [WAM030 - Employee Enquiry](#)
- [WAM035 - An Employee Update WAM](#)
- [WAM040 - Add dropdown lists for Department and Section](#)
- [WAM045 - A Dynamic Selector Dropdown list using a Select Field](#)
- [WAM050 - A Section Maintenance Application](#)
- [WAM055 - Using LANSAs Debug](#)
- [WAM060 - Employee Maintenance using Advanced Weblets](#)
- [WAM065 - Controlling List Output](#)
- [WAM070 - Hiding Techniques](#)
- [WAM075 - Using a Tree View Weblet](#)
- [WAM080 - Session Management](#)
- [WAM085 - Enhancing the User Interface](#)
- [WAM090 - Using a List Row Weblet](#)
- [WAM095 - LOB Data Types and Stream Files](#)
- [WAM100 - Using Cascading Style Sheets](#)
- [WAM105 - Create Your Own Weblet](#)
- [WAM110 - Create Your Own Layout Weblet](#)
- [WAM115 - Check in WAMs to IBM i](#)
- [WAM120 - Using the Menu Bar Weblet](#)

- [WAM125 - Define a Dynamic Menu](#)
- [WAM130 - Output a Web Page to a File](#)
- [WAM135 - Using the Google Static Maps API](#)

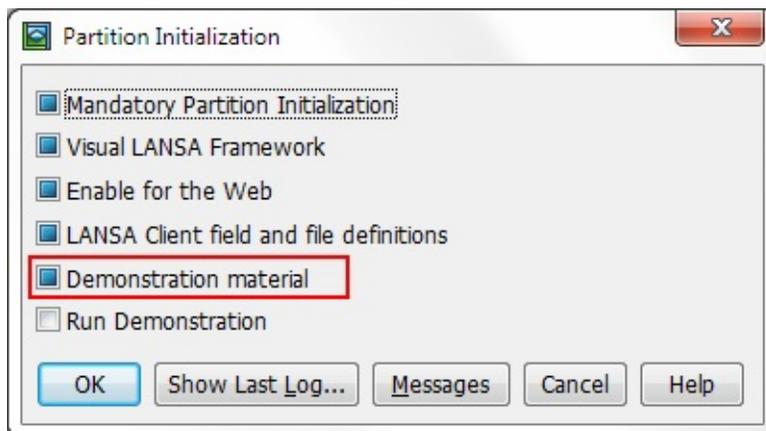
## Before You Begin

Some extra files are needed to complete these tutorials. Where these files are required, they are listed at the beginning of the tutorial. Before you begin, you can download the extra files from the documentation page of the LANSAs web site. Go to [Documentation - Current Version](#) and then look down the list for the Extra files in the Web Guide section as shown here:



The zip file includes a readme.txt file with instructions for what you need to do with these files, but the same instructions are also included at the beginning of each exercise in which you will use them.

You also need to ensure you have the Personnel Demonstration Files in the partition you will be using.



If you require instructions how to do this, go to [Partition Initialization](#).

## **What is a WAM?**

A WAM is a component that supports the web paradigm. A WAM outputs XML containing the fields and lists that are to be passed into its output web page. This web page is created via an XSLT transformation. The XSL reads the output XML and generates the XHTML. A web server associated with your web application then forwards this XHTML to the user's browser.

The XSLT transformation uses other LANSAXSL components, known as webllets. These webllets provide web components that can be added into a page and configured. They have properties which can be set both at design time and run time as required. Webllets include relatively simple components such as a push button or clickable image as well as much more complex components such as a grid, or tree view.

The web page layout is controlled by a layout webllet. A layout webllet is assigned when the WAM is compiled and may be modified both at design time and run time.

The WAM Design view provides a graphical editor that enables the developer to define the appearance and behavior of the web page that is output for a web routine. A webllet such as a push button can be dropped onto the page and then set up via its properties using the Details tab.

A WAM usually contains a number of web routines. Each web routine is a program entry point – it can be invoked via link on a web page for example. When a web routine ends, it outputs its fields and lists as an XML document.

Mapping of data into and out of a web routine is controlled by WEB\_MAP statements. A WEB\_MAP defines whether fields are mapped into or out of the web routine, or in both directions, via a FOR() parameter. Fields and lists are mapped via the WEB\_MAP FIELDS() parameter. Fields and lists may have display attributes that control whether the field is input capable (the default) or output only. Other field attributes such as \*private and \*hidden, may be used.

### **Some Example WEB\_MAP Statements**

```
WEB_MAP FOR(*OUTPUT) FIELDS(#EMPNO)
```

Field EMPNO is mapped out of this Web Routine. EMPNO is input capable on the page.

```
WEB_MAP FOR(*INPUT) FIELDS(#EMPNO)
```

Field EMPNO is mapped into this Web Routine. EMPNO is not mapped out of this Web Routine to the page.

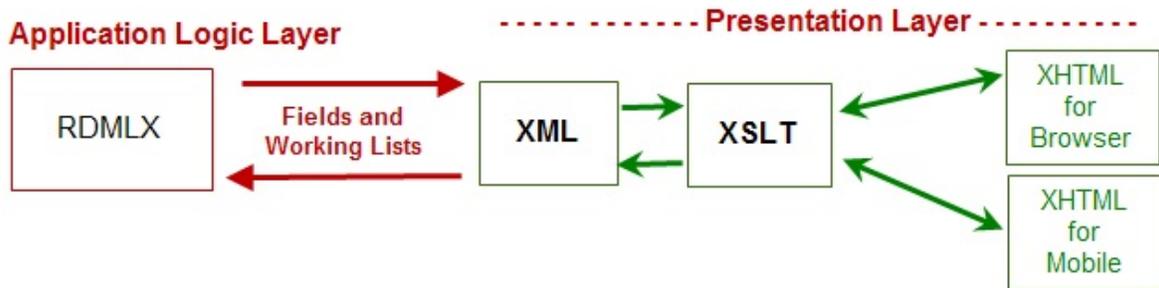
```
WEB_MAP FOR(*BOTH) FIELDS((#EMPNO *OUTPUT)
#SURNAME #GIVENAME)
```

Fields are mapped into and out of this Web Routine. Field EMPNO is output on the page and therefore cannot be mapped into the next Web Routine.

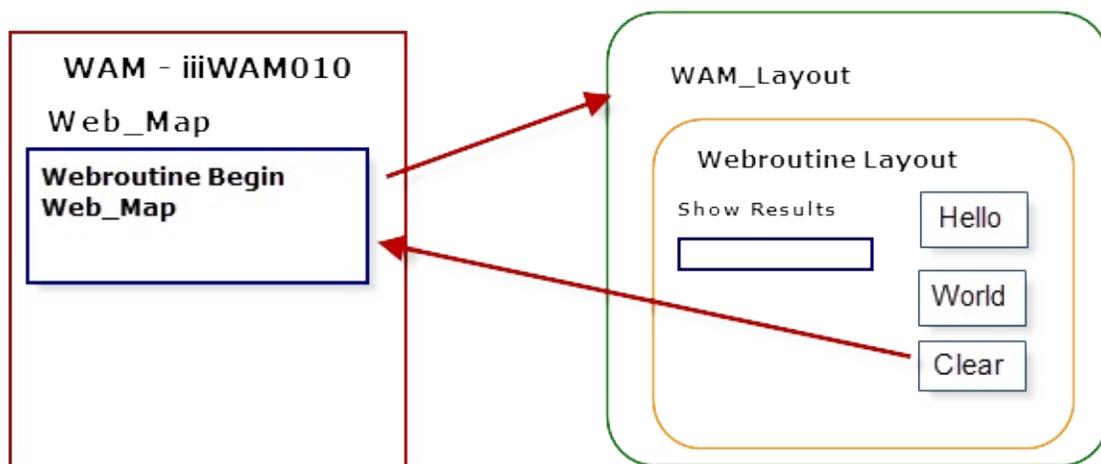
```
WEB_MAP FOR(*BOTH) FIELDS((#STDREENTRY *HIDDEN))
```

Field STDREENTRY is mapped into and out of this Web Routine. STDREENTRY is a hidden field defined in the XHTML but not shown on the web page.

A WAM application should be considered as having two layers:



## WAM Architecture



- A WAM contains one or more WebRoutines
- A WebRoutine usually outputs a web page
- Web\_Maps define fields and lists that may be mapped into and out of the WebRoutine

- Web\_Maps defined at WAM level apply to all WebRoutines
- A WAM adopts a standard layout
- All WebRoutine layouts share the wam\_layout (initially)
- The WebRoutine web page contains fields and lists that are mapped for \*output or \*both
- Developer completes WebRoutine page design using a graphical editor (the Design View)
- Weblets provide XSL that add web components such as push buttons to the web page
- Weblets are set up (programmed) using a property sheet on the Details tab.

## Stateless Programming

One of the key points to understand about WAMs is that they are stateless. In fact, any internet-based application is stateless. What this means is that when a WAM is executed from the Presentation Layer, it runs (a job is initiated on the server), produces some output (a web page), and then ends (the job on the server ends and control is transferred back to the browser).

The job starting and ending, to all intents and purposes, is a "transaction". Any data that needs to be maintained for the user's web "session", i.e. span multiple transactions, must be kept somewhere. As you complete the following simple example WAM, you will begin to see how a web application needs to be designed to handle this "stateless" model.

## Using Long Names

Components can be defined using *Long Names*. This is an optional setting at *Partition* level. When you create a WAM, form or a reusable part using a Long Name, Visual LANSA will assign a unique *Identifier* (this is up to 10 characters in length). As you are creating a new component, you may choose to assign your own *Identifier*, which provides complete control to a team of developers, of both Long Names and Identifiers.

See the topic *LANSA Object Names* in the *Technical Reference* guide for further information.

This workshop assumes that Long Names are enabled.

## WAM005 - Create Your First WAM

### Objectives

- To create your first Web Application Module (WAM) and become familiar with the LANSAs Editor's *Design* view.
- To create a WebRoutine in your Web Application Module. You will create a WebRoutine that will use different weblets to call itself. It will display a different message depending on which weblet is used.
- To understand the basics of how the WEB\_MAP command is used within a WebRoutine (covered in much more detail in [WAM010 - Using WEB\\_MAPs](#)).
- To introduce you to re-entrant programming.
- To compile a Web application that was created using a WAM.
- To understand the compile/generate cycle of WAM development.

To achieve these objectives, you will complete the following steps:

[Step 1. Start Visual LANSAs](#)

[Step 2. Create a WAM](#)

[Step 3. Create the ReentryTest WebRoutine](#)

[Step 4. Compile the WAM](#)

[Step 5. Open the Design view](#)

[Step 6. Editing](#)

[Step 7. Use a Weblet](#)

[Step 8. Make STDREENTRY visible for testing](#)

[Step 9. Test the WAM](#)

[Step 10. Hide STDREENTRY](#)

[Summary](#)

### Before you Begin

In order to complete this exercise, the demonstration Personnel System must have been installed in this partition.

In order to create a WAM:

- You must use an RDMLX Enabled Development partition.
- Field Types such as Date and Time must also be enabled in the partition.

- The partition must be web enabled.
- You must have a valid LANSA for the Web license to develop with WAMs.

For details, refer to the *LANSA for i User Guide* and the *Web Administrator Guide*.

## **Step 1. Start Visual LANSA**

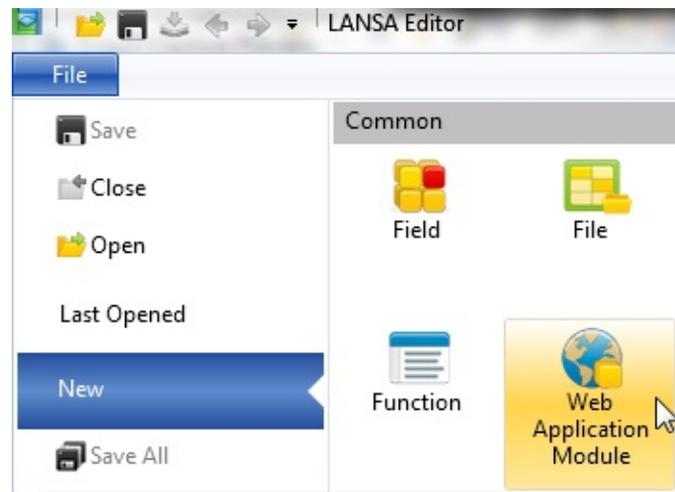
WAMs are created using Visual LANSA and may be deployed to a variety of servers such as Windows and IBM i. In this step, you will log on to Visual LANSA.

1. Start the Visual LANSA Development Environment and log on to the DEM partition (recommended) or any other partition being used for training. The partition must contain the Personnel System files.

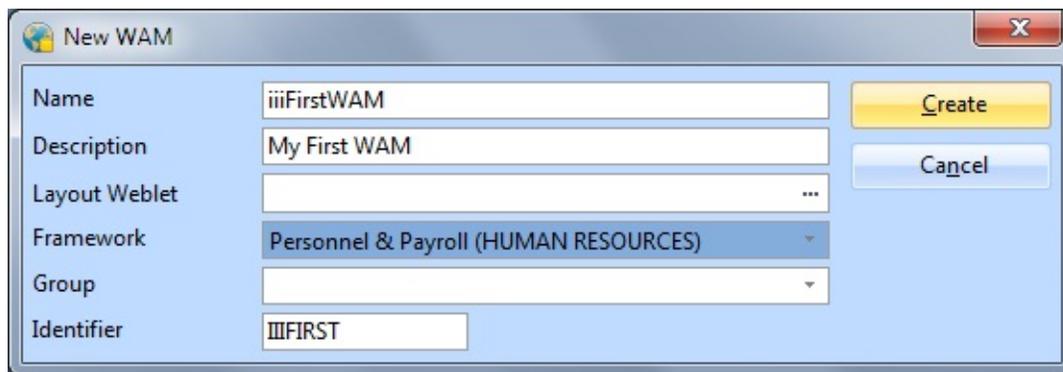
## Step 2. Create a WAM

In this step, you will create a Web Application Module that will eventually contain the RDMLX code for your First WAM.

1. In the LANSAs Editor window, select File, then New and *Web Application Module*:



The *New WAM* dialog will appear:

A screenshot of the 'New WAM' dialog box. The dialog has a title bar with a globe icon and the text 'New WAM'. It contains several input fields and buttons. The 'Name' field contains 'iiiFirstWAM', the 'Description' field contains 'My First WAM', the 'Layout Weblet' field is empty with a dropdown arrow, the 'Framework' dropdown is set to 'Personnel & Payroll (HUMAN RESOURCES)', the 'Group' dropdown is empty, and the 'Identifier' field contains 'IIIIFIRST'. There are 'Create' and 'Cancel' buttons on the right side.

2. In the *New WAM* dialog box:
  - a. Enter a *Name* of **iiiFirstWAM** (where **iii** are your initials).
  - b. Enter a *Description* of **My First WAM**.
  - c. Leave the Layout Weblet field blank.
  - d. Select a Framework Personnel & Payroll.
  - e. Click the *Create* button to create the new WAM.

**Note:**

When you leave the Layout Weblet field blank, your WAM will automatically generate a WAM layout based on Theme Layout #1 – One Column (d\_themelet1\_1col.xsl). You will create and use your own layout using a wizard in a later exercise.

Frameworks allow related components to be grouped together. The framework assigned to a component may be changed at any time. The Personnel & Payroll framework has been selected for this new WAM. The new WAM will be given a ComponentFramework property HUMAN RESOURCES-Personnel.

3. The *New WebRoutine* dialog will appear. Deselect the *Show this dialog . . .* option and select *Cancel*. You will create your WebRoutines manually in the editor. You will use this dialog in a later exercise.

Enter a Webroutine name and choose Create to begin designing your Web Page

Webroutine details

Name

Description

Service Name

Response \*DEFAULT - return a Web Page

Web Page design details

Generate XSL	Open Design	Provider	Technology Service
<input type="checkbox"/>	<input type="checkbox"/>	LANSA	JQMOBILE
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	LANSA	XHTML

Show this dialog when Web Application Module with no Webroutines is opened

4. The LANSAs Editor will now display the WAM's RDMLX code. At this stage, it will not contain any WebRoutines:

```
Design | Source | Repository Details | Cross References |
* *****
*
* COMPONENT:  STD_WAM
*
* *****
Function Options(*DIRECT)
☐ Begin_Com Role(*EXTENDS #PRIM_WAM)

End_Com
```

### Step 3. Create the ReentryTest WebRoutine

In this step, you will use the LANSAs Editor to create the code for a new WebRoutine that will give you an understanding of Weblets and re-entrant programming. You will use a WEB\_MAP statement to specify the fields that are passed in and out of the WebRoutine. Your initial code looks like the following:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)
```

```
End_Com
```

1. Immediately following the BEGIN\_COM, insert the following RDMLX code to create a WebRoutine named *ReentryTest*:

```
WebRoutine Name(ReentryTest) Desc('Test WAM')
Endroutine
```

2. This WebRoutine will require three fields that are both incoming and outgoing.
  - a. STDREENTRY will be used to control the functionality of the WebRoutine each time it is called.
  - b. GIVENAME and SURNAME are to demonstrate the different display modes for fields on the page.
  - c. Add the following WEB\_MAP statement to the WebRoutine:

```
Web_Map For(*BOTH) Fields((#Givename *input) (#Surname *output)
(#Stdreentry *hidden))
```

The **STDREENTRY** field will be used to identify the execution state.

The **STDREENTRY** field is qualified as hidden so as not to be visible on the page.

The **GIVENAME** field will accept input.

The **SURNAME** field will be output only.

3. Set up a **CASE loop using** the field **STDREENTRY** to display different messages.

```
Case Of_Field(#Stdreentry)
When Value_Is(= A)
```

```

Message Msgtxt('Push Button was pressed')
When Value_Is(= B)
Message Msgtxt('Hyperlink was clicked')
When Value_Is(= C)
Message Msgtxt('Check Box was changed')
When Value_Is(= D)
Message Msgtxt('Radio Button was selected')
When Value_Is(= E)
Message Msgtxt('Dropdown was changed')
Otherwise
Message Msgtxt('Page loaded normally')
Endcase

```

4. Set **GIVENAME** and **SURNAME** to your name.

```

#Givename := 'your first name'
#Surname := 'your last name'

```

Your finished WebRoutine should appear as follows:

```

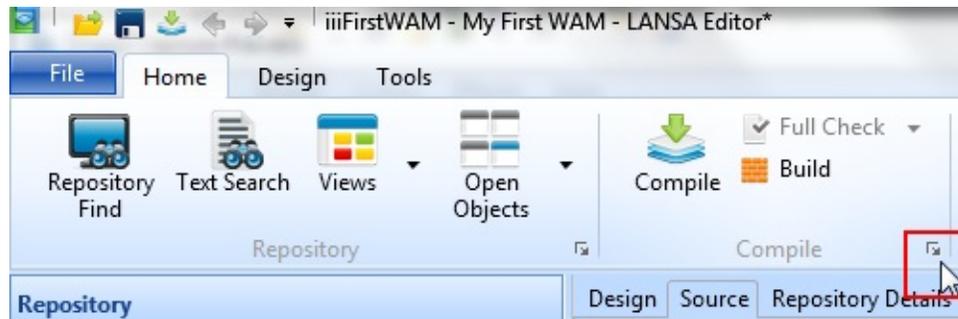
Webroutine Name(ReentryTest) Desc('Test WAM')
Web_Map For(*BOTH) Fields((#Givename *input) (#Surname *output)
(#Stdreentry *hidden))
Case Of_Field(#Stdreentry)
When Value_Is(= A)
Message Msgtxt('Push Button was pressed')
When Value_Is(= B)
Message Msgtxt('Hyperlink was clicked')
When Value_Is(= C)
Message Msgtxt('Check Box was changed')
When Value_Is(= D)
Message Msgtxt('Radio Button was selected')
When Value_Is(= E)
Message Msgtxt('Dropdown was changed')
Otherwise
Message Msgtxt('Page loaded normally')
Endcase
#Givename := 'your first name'
#Surname := 'your last name'
Endroutine

```

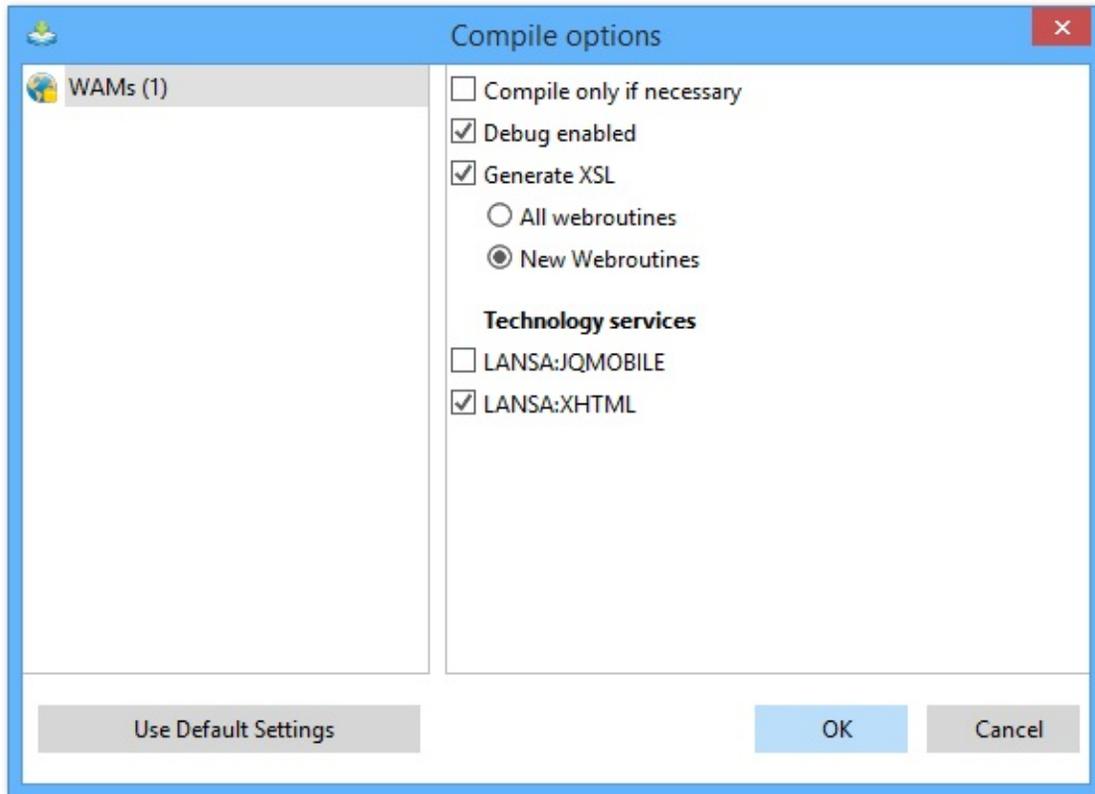
## Step 4. Compile the WAM

In this step, you will learn how to compile the WAM.

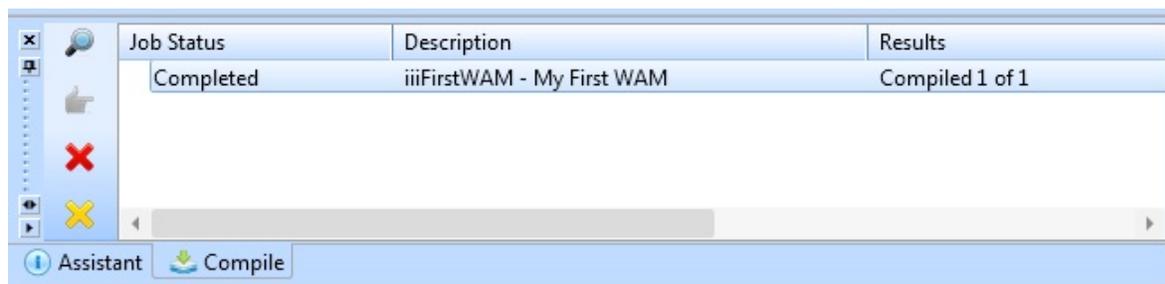
1. Make sure you have the **iiiFirstWAM** (where **iii** are your initials) open in the LANSa Editor.
2. Click on the dialog box launcher in the *Compile* group on the ribbon to display the Compile options dialog:



3. On the *Compile options* dialog, ensure that only the *Technology Services*, *LANSa:XHTML* is selected. These tutorials will be generating web applications for the browser only.  
These options will be retained for future compiles.



4. Click OK to compile the WAM. The status of the compile is displayed in the Compile tab:



The compile will generate a browser web page design (XSL) for you.

The generator performed the following steps:

Generated your WAM-specific layout Weblet, called **iiiFirstWAM\_layout**. You can see this in the Repository under Weblets in the Web View.

Generated the WebRoutine web page design using the layout **FirstWAM\_layout**.

Placed the WebRoutine description as a heading on the web page.

Included a messages Weblet on the page.

Placed all fields mapped FOR(\*OUTPUT) or FOR(\*BOTH) ) and their captions a table on the web page.

Inserted a hidden DIV at the bottom of the page. In the *Design* view this has the label "Hidden Content" shown. You can add anything that you want to be hidden here and at run time, it will not be displayed on the page regardless of its display mode.

## Step 5. Open the Design view

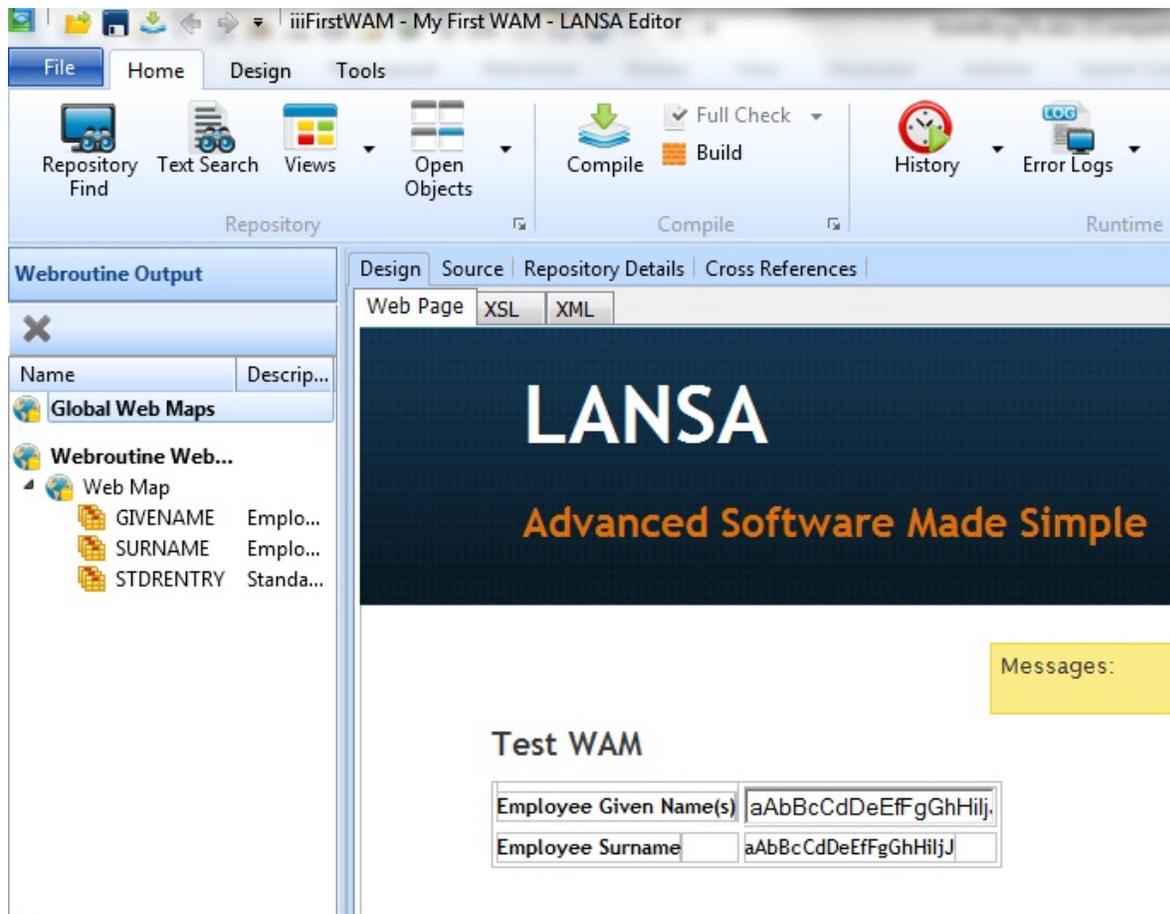
After creating the WebRoutine RDMLX within the LANSAs Editor, you can edit the web page design for your WebRoutines. In this step, you will select the *ReentryTest* WebRoutine created in Step 1 and open the Design view.

1. After the compile has completed successfully, open the *ReentryTest* WebRoutine in the Design view.
  - a. Click the Open Design  button to open WebRoutine *ReentryTest* in the Design view.

```
* *****  
*  
* COMPONENT: STD_WAM  
*  
* *****  
Function Options(*DIRECT)  
[-] Begin_Com Role(*EXTENDS #PRIM_WAM)  
  [-] Webroutine Name(ReentryTest) Desc('Test WAM')   
    Web_Map For(*BOTH) Fields((#Givenname *input) (#Surname  
    [-] Case Of_Field(#Stdreentry)  
      When Value_Is(= A)
```

- b. The *Design view* will open for the WebRoutine.

The Design view will appear something like the following:



Once a WebRoutine has been compiled, a web page design with a WAM-specific layout (that you can modify) and field entry controls are generated and can be edited in the *Design* view.

**Note:** Because you now have a layout Weblet that is only used by the **iiiFirstWAM** WAM, you can open it and edit it without affecting the layout of any other WAMs.

The left hand pane includes these views used for working with WAMs:

**Outline** - shows fields and lists, as well as other elements contained in the XSL ge currently being edited.

**Details** - allows the editing of attributes of selected XSL and HTML elements, well as Weblet parameter values.

**WebRoutine Output** - shows all the fields and lists available for drag and drop to the web page. These are all the fields and lists specified on the WebRoutine's EB\_MAP FOR(\*OUTPUT) or WEB\_MAP FOR(\*BOTH) statements in RDML.

The right hand Design tab contains the following Views:

**Web Page** - allows WYSIWYG editing of the web page.

**XSL** - allows text editing of the XSL source.

**XML** - allows text editing of the input XML source.

The bottom pane contains the following View:

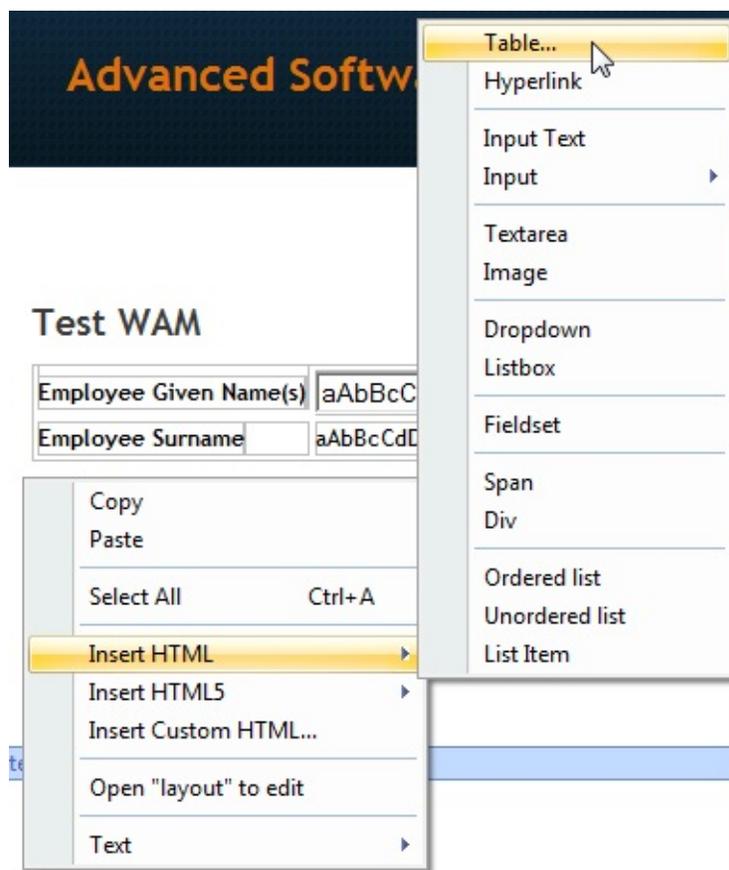
**Web Designs** - lists all WebRoutines for the current WAM open in the editor. Every WebRoutine on the tab represents a web page design using certain technology in certain a Language.



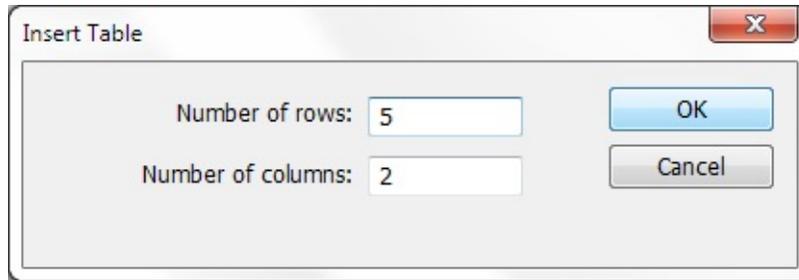
## Step 6. Editing

In this step, you will perform some simple editing tasks in the Design view and preview your changes in order to become familiar with the Editor. You will add a table, add text and Weblets to the table and set the Weblet's parameters.

1. Make sure you are in the *Design View* by clicking on the *Design* tab. To insert the table into the *Design View*:
  - a. Right-click the mouse in the middle of the page, under the table that was generated from the compile, to open the context menu:
  - b. Select *Insert HTML* from the context menu and choose the *Table* sub menu item:



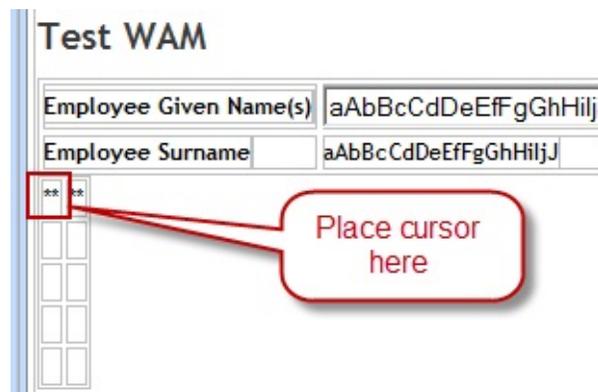
- c. Create a table with 5 rows and 2 columns in the *Insert Table* dialog box. Press *OK*.



You will now see an HTML table has been added in the *Design View*.

Notice that the top row contains asterisks (that is "\*\*\*"). These characters are inserted so that you can easily access the table cells. Without the asterisks the table cells would have no visible width. After editing the table, you can simply delete the asterisks.

2. Place the cursor into the top, leftmost cell of the new table in the *Design View*.



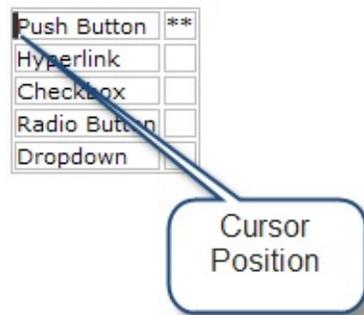
3. Type the text "Push Button" into the first cell of the table. This column will be used as labels for the weblets that will be added to the second column.
4. Now that this column has an entry you can delete the placeholder "\*\*\*" characters from the first table cell by placing the cursor next to them and using the *Delete* button.
5. Insert the names of the other Weblet labels in the four remaining blank rows of the left column: "Hyperlink", "Check Box", "Radio Group" and "Dropdown". There is no need to include the quotation marks.

Your *Design View* will now appear something like this:

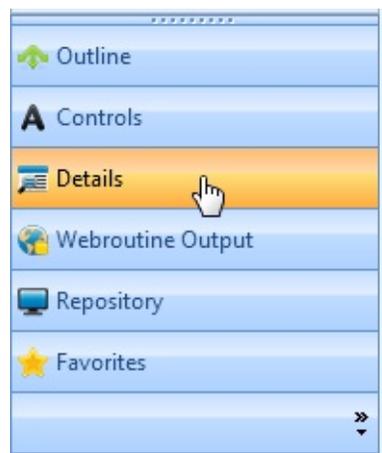
### Test WAM

Employee Given Name(s)	aAbBcCdDeEfGhHilj.
Employee Surname	aAbBcCdDeEfGhHiljJ
Push Button	**
Hyperlink	
Check Box	
Radio Button	
Dropdown	

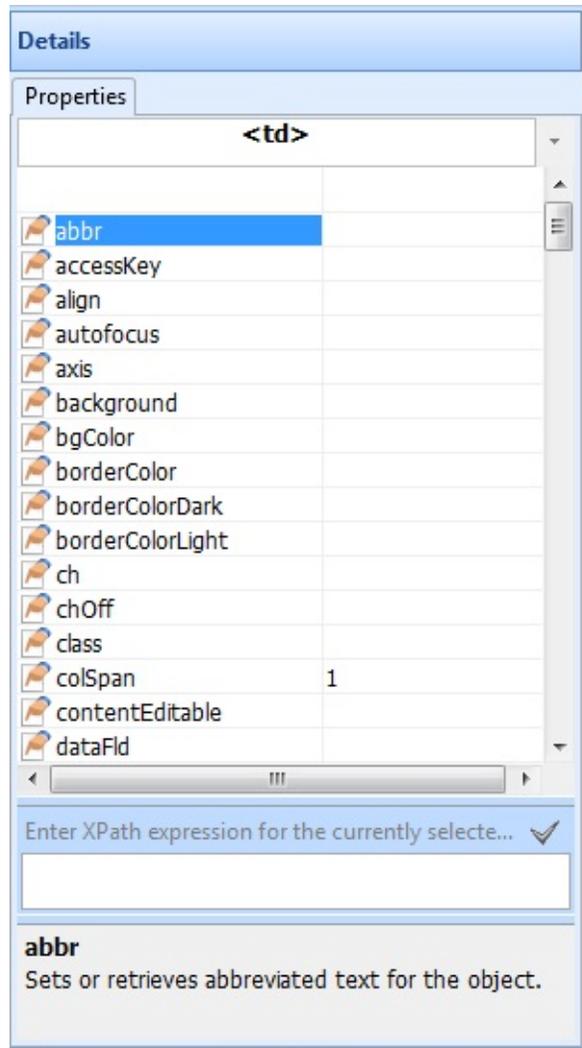
- In the *Design View*, select the top left cell again by clicking in it. (That is, the table cell containing the text "Push Button".) There should be no "Grips" around anything at this point and the cursor should be blinking inside the cell.



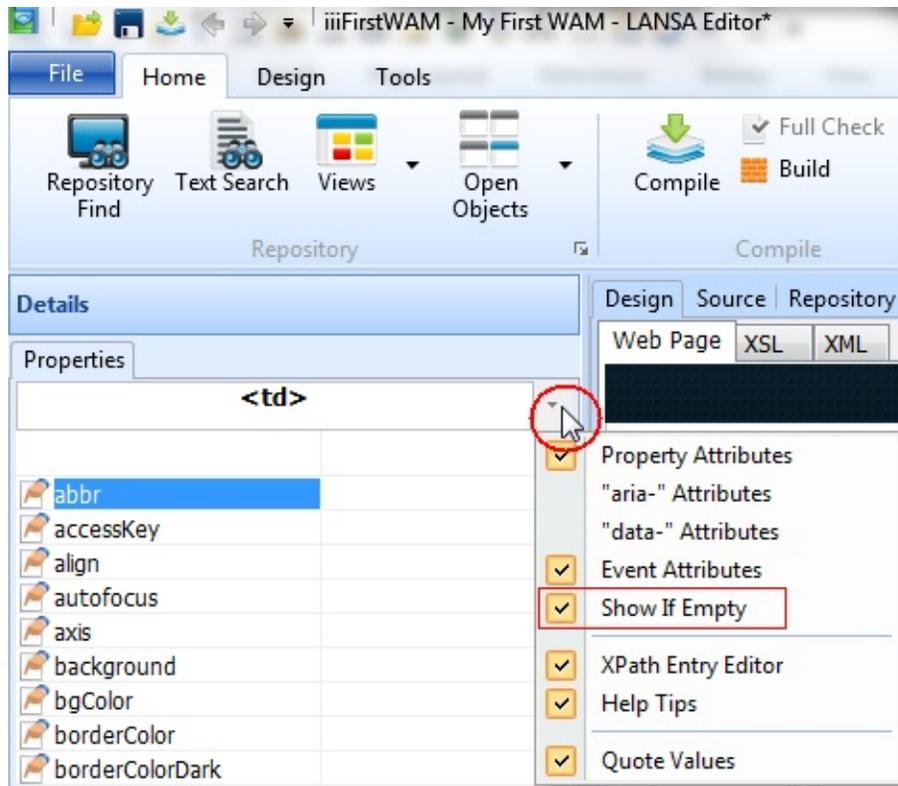
- Select the *Details View* in the left pane:



You will now see all the properties of the table cell:

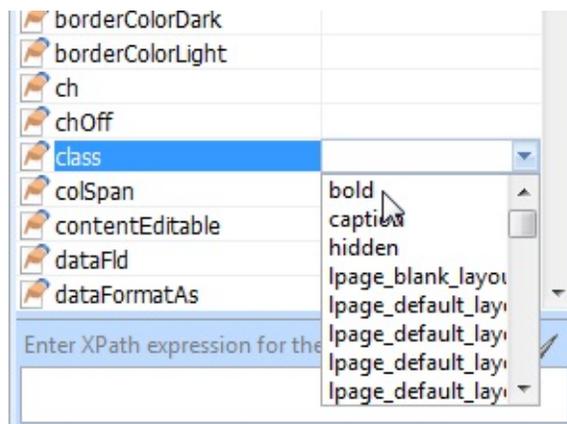


8. Click the *Menu* button on the *Details* tab and make sure the *Show If Empty* option is selected and that the *"aria-" Attributes* are not selected.



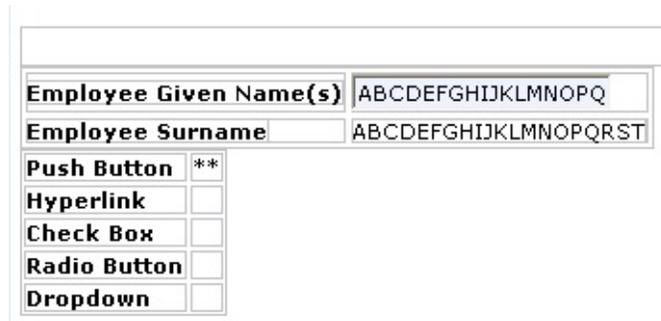
The *Show If Empty* option displays properties for the cell which currently do not have a value.

9. In the *Details View*, locate the *class* property.
  - a. Click in the *class value* column and use the dropdown list to set the property to **bold** and press *Enter*:



The text will immediately become bold. What you have done is applied one of the caption styles provided with the Cascading Style Sheet (CSS) shipped with the LANS product. The default style is not bold.

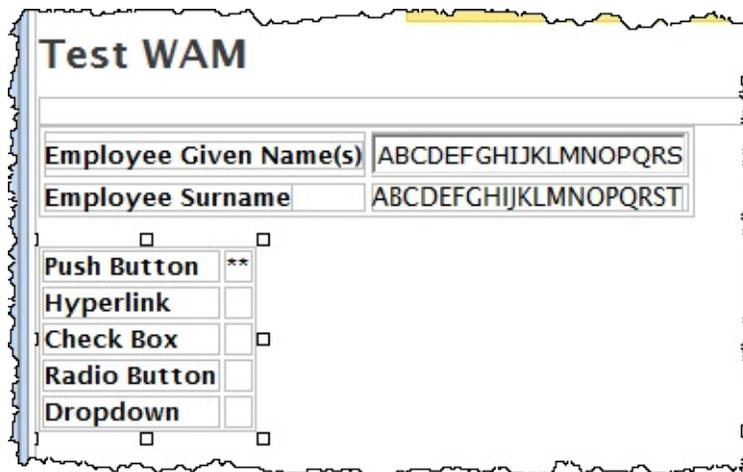
b. Set the *class* property to **bold** for the remaining cells in the left column. The *Design* view should appear something like the following:



10. Click the *Save* button to save your changes to *ReentryTest*:



11. Select the new table by clicking on one of its corners:



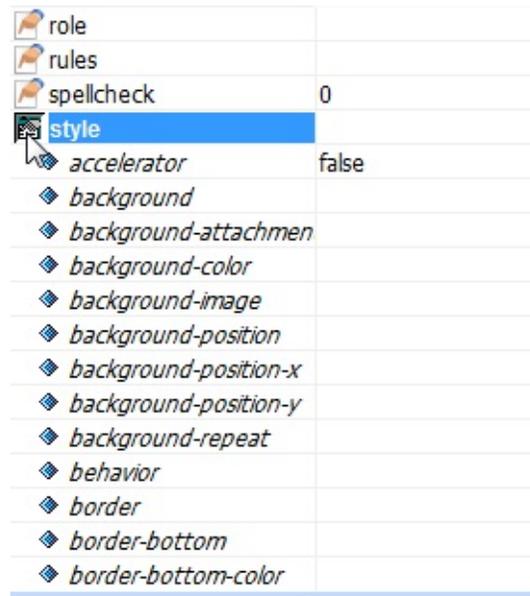
A selected web page element is displayed with "handles" as shown in the screen picture.

12. Scroll down to the *style* property. Style is a composite property consisting of other properties.

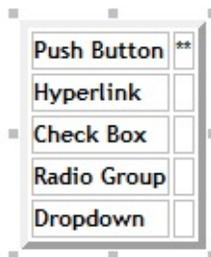
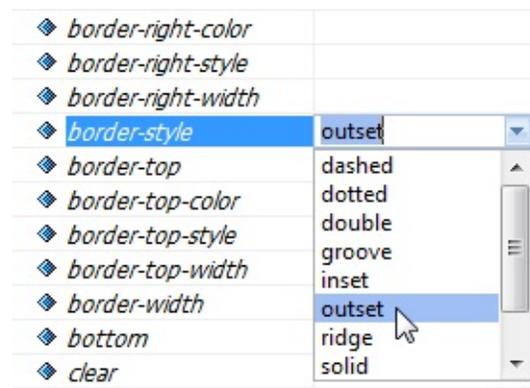


13. Click on the icon to the left of the property to expand the style's individual

properties. You can modify these properties directly.



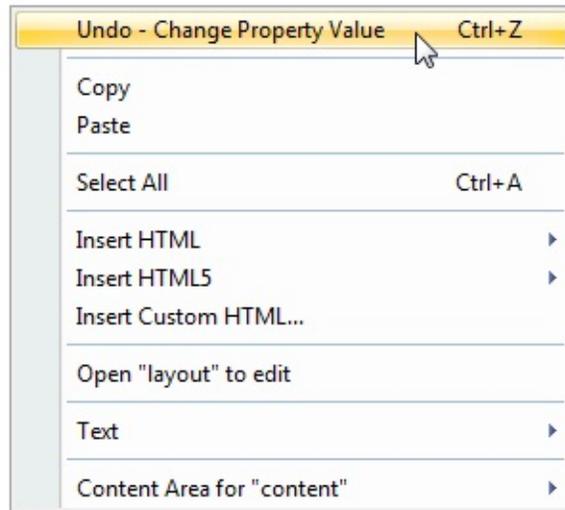
14. Scroll down to style's *border-style* property. Select **outset** from the dropdown list. Click on another property and you should immediately see the border of the cell change to an outset appearance.



15. Experiment by changing other properties, such as the background color, or inserting other HTML elements into the page. Using styles in this way sets an

inline style for the HTML element that you are editing. You will usually want to control styles via a Cascading Style Sheet (CSS). Refer to exercise [WAM100 - Using Cascading Style Sheets](#) for more information about CSSs.

16. Right-click the web page design and select *Undo* from the context menu to remove the changes made in 14.

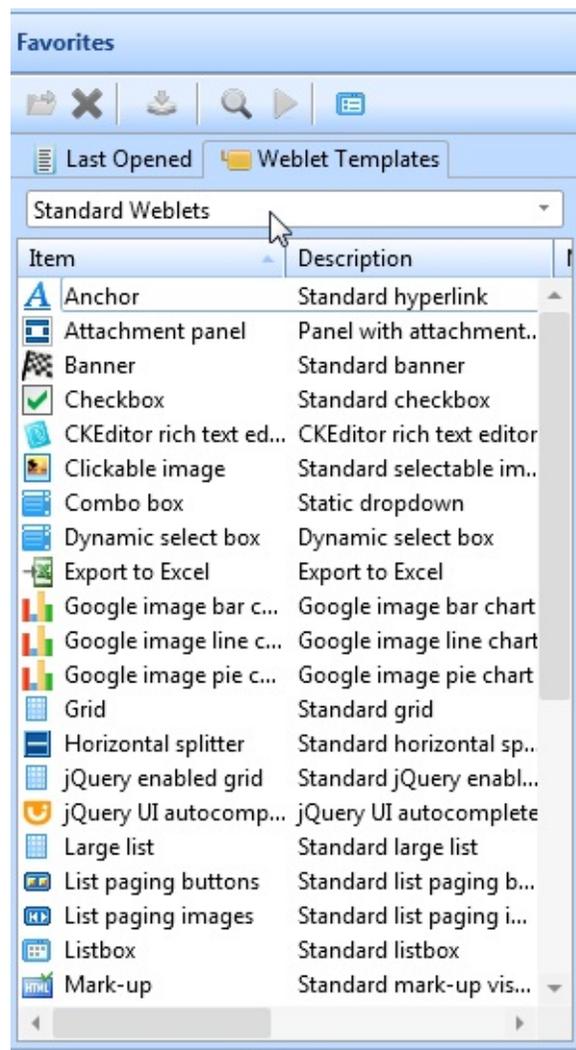


Alternatively, you could also have used the **Ctrl+Z** keys to undo.

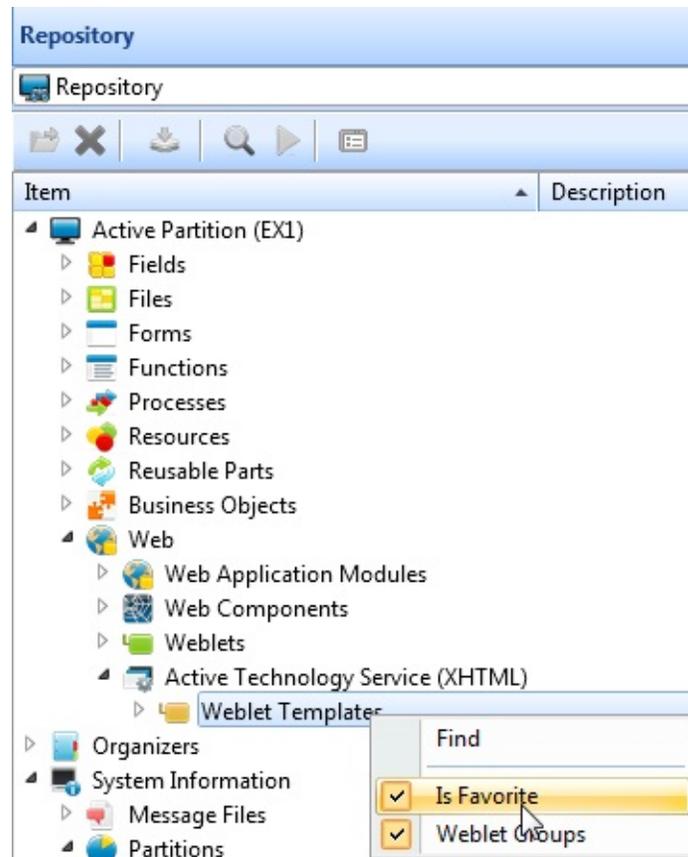
## Step 7. Use a Weblet

In this step, you will learn how to drag and drop Weblets onto the web page. The Weblets will each be configured to call the WebRoutine with a different reentry value.

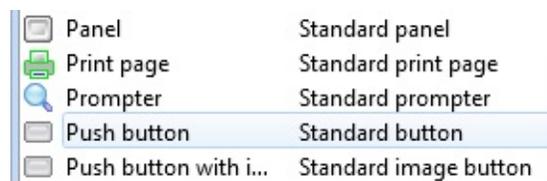
1. Make sure *ReentryTest* is open in the Design view.
2. Select the Favorites/*Weblet Templates* tab in the left pane to see a list of all the Weblets in your repository. (Make sure that Standard Weblets is specified in the dropdown list so that all standard Weblets are listed.)



**Hint:** If *Weblet Templates* is not shown on your *Favorites* tab, select the *Repository* tab and expand *Web / Active Technology* and then use the right mouse menu to make *Weblet Templates* a *Favorite*.

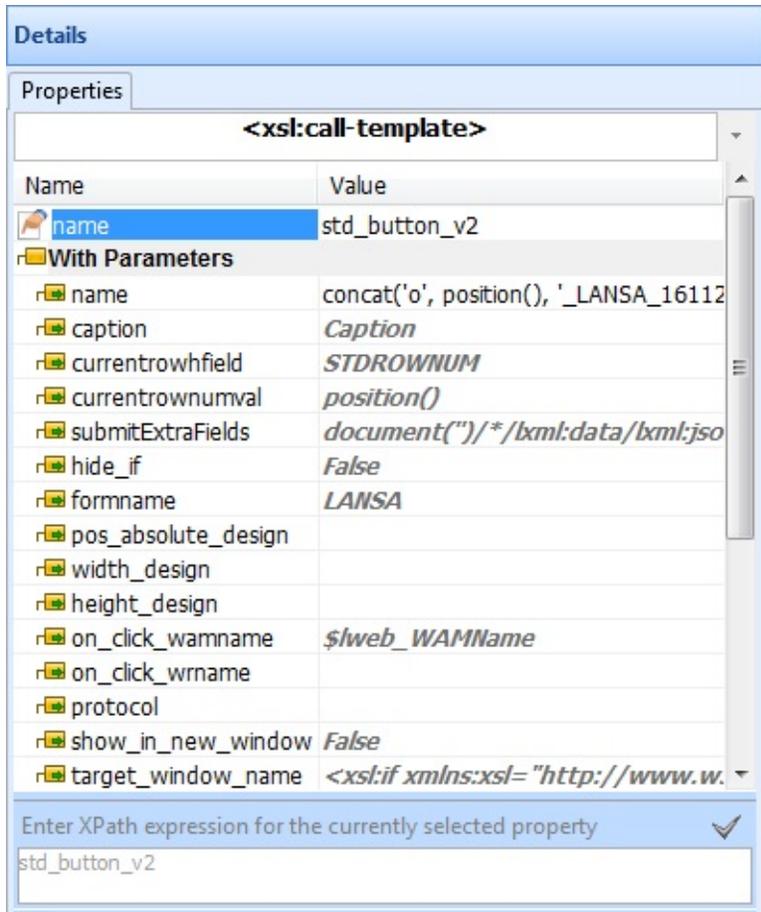


3. Select the *Push button* Weblet from the list and drag and drop it into the cell that is to the right of the "Push Button" caption in the table in *Design View*.

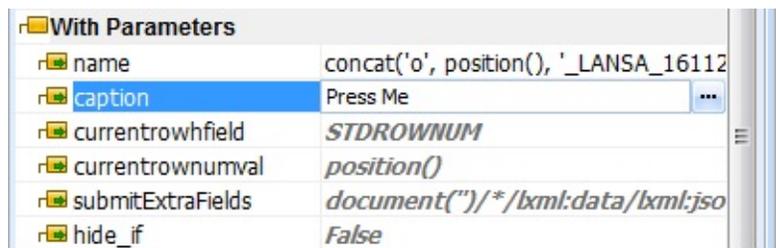


Push Button	* Caption *
Hyperlink	
Check Box	
Radio Group	
Dropdown	

4. Select the button and display the *Details* tab in the left pane to display the Weblet's properties. The italicized and gray values indicate default values.

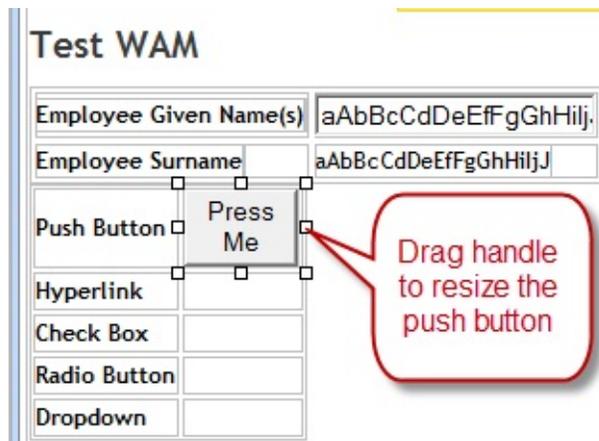


In the *Details View*, click in the value column of the *caption* property and type in **Press Me**.

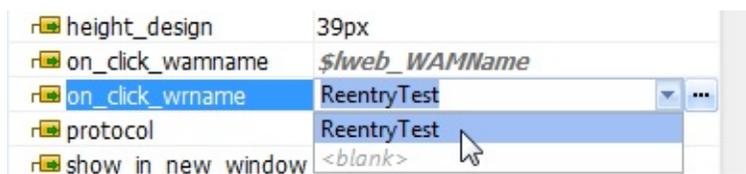


The button's caption will immediately change to **Press Me**.

If necessary, drag the right handle on the push button so that the caption is displayed as a single line.



Set the `on_click_wname` property to **ReentryTest** by selecting this value from the value dropdown.

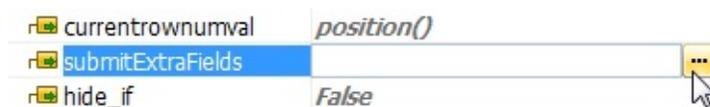


**Note:**

You should always use the dropdown list to select this property rather than type it in, as spelling mistakes or typing errors are one of the biggest causes of problems at design time.

Always complete the `on_click_wname` property before the next step. The *Design of...* dialog uses the WebRoutine defined in the `on_click_wname` property to populate the list of mapped fields in the *Fields* dropdown list.

- c. Click on the `submitExtraFields` property and then click the *Ellipsis* button to open the *Design of ....* dialog.





- d. Select the field STDREENTRY in the left hand *Name* field.
- e. Enter a value of **A**, in the *Value* column, leaving the *Literal* checkbox selected.
- f. Click *OK* to confirm your changes and close this dialog.

**Note:** The fields shown in the dropdown for the Name column in the *Design of submitExtraFields Property* dialog are the fields mapped for \*INPUT or \*BOTH for the WebRoutine which is defined as the *on\_click\_wrname* property for this weblet.

In *Value*, if you specify a:

- Literal value, you are returning a fixed, hard coded value for this field when this button is clicked.
  - Field, you are returning the field value when this button is clicked.
5. In the *Design View*, click on the table cell where the Push Button weblet was dropped. Again, make sure that you select the cell and not the button. Delete the asterisks in the cell by placing the cursor in the cell next to them and using the backspace key.

**Hint:** With the push button selected you can use the *arrow keys* to move left or right into the table cell.

6. In the *Details View*, set the *align* property of the cell to **right**.



7. Now add the remaining Weblets by selecting them in the *Weblets Templates View* and dragging them into the appropriate right hand cell based on the left hand cell label. The Weblets you will use are the Anchor, Check Box, Radio

## Group and Combo Box.

Push Button	Press Me
Hyperlink	<a href="#">Hyperlink</a>
Check Box	<input type="checkbox"/> Caption
Radio Group	<input type="radio"/> Radio 1 <input type="radio"/> Radio 2
Dropdown	Item 1 ▼

8. All of the remaining weblets have default of STDREENTRY for their *reentryfield* property. Set the *reentryvalue* property for each of these Weblets. Check the RDMLX to see what value should be used for each weblet. For example:

```
When Value_Is(= B)  
Message Msgtxt("Hyperlink was clicked.")
```

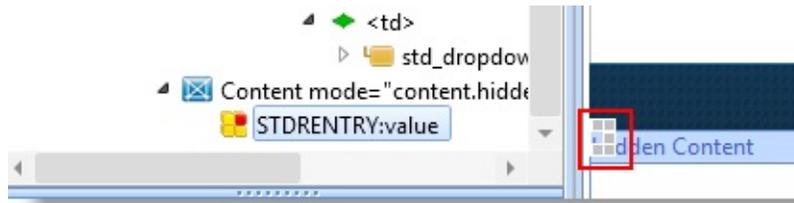
9. Set the *on\_click\_wrname* property for the anchor, check box and radio group to **ReentryTest**.
10. Set the *on\_change\_wrname* property for the combo box to ReentryTest.  
Notice that you are not required to change the *on\_click\_wamname* or *on\_change\_wamname*. By default these values will be the WAM name that initiated the web page.
11. Set the *value* property for the anchor to **Click Me**.
12. Set the *caption* property for the check box to **Click Me**.
13. Set the *align* property for all of the cells containing weblets to **right**.

Your page should now appear as follows:

Push Button	Press Me
Hyperlink	<a href="#">Click Me</a>
Check Box	<input type="checkbox"/> Click Me
Radio Group	<input type="radio"/> Radio 1 <input type="radio"/> Radio 2
Dropdown	Item 1 ▼



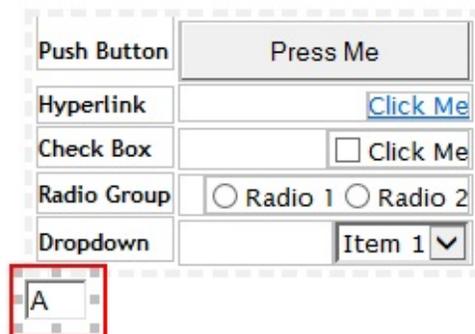
page, with "Grips" around it.



- c. Right click on the highlighted field in the Hidden Content area and select *Replace with Input Field*. Be careful not to deselect STDRENTY until you have changed it into an input field. If it does happen to become deselected and you can no longer see it, select it again in the Outline tab.



2. Select the field STDRENTY in the *Hidden Content* area and use the right mouse menu to *Cut* it. Click on the main page area below the new table and use the right mouse menu to *Paste* the field here. Your design should now look like the following:

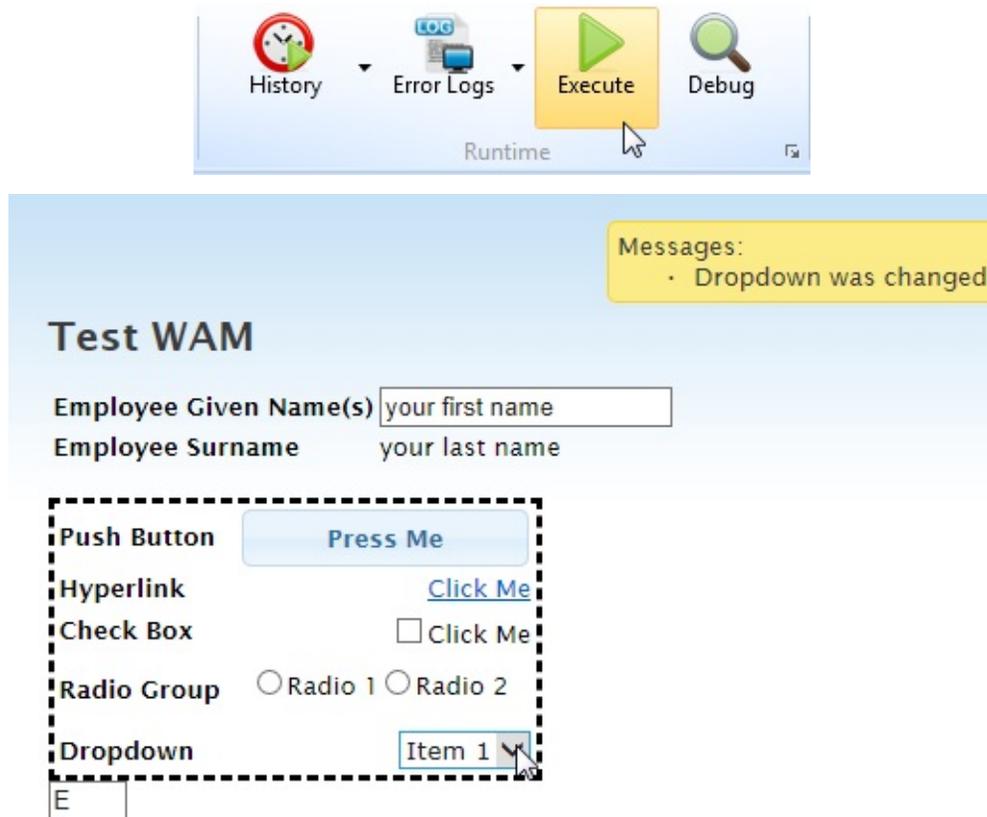


3. Save the changes to the layout.

## Step 9. Test the WAM

In this step, you will test the WAM.

1. *ReentryTest* should still be open in the *Design* view. If not, open it.
2. Click the Execute button on the ribbon to run your WAM *in the Web Browser*.



3. Use each Weblet and see that the appropriate messages are given. If they are not, check the `STDREENTRY` to see what its value is and make sure the Weblet itself has the correct `reentryvalue` set. Make any necessary changes and test again.
4. When you are satisfied that the page is functioning correctly, close the browser window.

## Step 10. Hide STDREENTRY

In [Step 9. Test the WAM](#), you have confirmed that the WAM is functioning properly, so you don't need STDREENTRY to be visible for testing. You will hide it again in this step.

1. Open *ReentryTest* in the Design view.
2. Drag STDREENTRY back into the Hidden Content area at the bottom of the web page.
3. At run time it will now be hidden, but if you would like to make it hidden in the *Design View* also, right click on the field and select *Change to Hidden Field*.
4. Save the changes to the layout.

## Summary

### Important Observations

- A WAM layout is created using *Theme Layout Weblet #1 – One Column* by default if a layout is not selected on the Create WAM dialog.
- If a WAM has not been compiled, if a WebRoutine is opened in the Design View, the web page will be generated using *std\_themelet1\_1col* weblet and the fields and lists mapped for output in the WebRoutine.

**Note:** In this case the web page will not be re-generated when the WAM is compiled unless the *Generate XSL – All WebRoutines* option is selected. There are more details on this subject in later exercises.

- Once a WAM is compiled, a web page with a WAM specific layout (that you can modify) and field entry controls is generated for each WebRoutine that can be edited in the Design view. You can drag and drop fields and lists marked FOR(\*OUTPUT) or FOR(\*BOTH) onto your page at any time from the Design view's *WebRoutine Output* tab.
- Weblets are reusable XSL snippets or components that can be plugged into either your WebRoutine page or other Weblets. There are standard Weblets that encapsulate HTML buttons, hyperlinks, menu items, radio buttons, dropdown lists, etc.
- When viewing a list of available Weblets in *Weblet Templates view*, you will notice a Weblet with a Description of *Push button will also have a Details column showing std\_button\_v2*. *Std\_button\_v2.xsl* is an XSL document consisting of one or more `<xsl:template>` element.
- The *Compile options* dialog, lets you choose whether or not to re-generate the XSL for existing WebRoutines. By default, a compile will always generate XSL for new WebRoutines ONLY. That is, the web page design for each new WebRoutine is generated automatically.
- The generated web page includes all fields and LISTS in your WEB\_MAPs marked FOR(\*OUTPUT) or FOR(\*BOTH). Fields mapped with a FOR(\*INPUT) keyword, are incoming fields to the WebRoutine. They are not available when creating the web page output by this WebRoutine.
- The compile generates a WAM-specific layout Weblet. Its name is WAM *Identifier* followed by *\_layout* (e.g. *iiifirst\_layout*) where **iiifirst** is the WAM's *Identifier*.

- The generated web page also has a *Messages (std\_messages)* Weblet on the page, a WebRoutine description in a heading and a *Hidden Content* DIV where you may place objects that you do not want to be displayed on the page.

## **Tips & Techniques**

- You can compile from either the LANSAs Editor directly, or from the Repository or Last Opened tabs.
- It is not necessary to do a compile to generate the web page, since a build also generates it, but without doing a full compile.
- When compiling, the *Generate XSL* option always defaults to the **New WebRoutines** option.
- The *Generate XSL All WebRoutines* option will generate, or re-generate the XSL for ALL WebRoutines in the WAM and will lose any changes made to the web pages in the Design view.
- You can make a hidden field visible for debugging purposes.

## **What I Should Know**

- How to create a Web Application Module (WAM).
- How to define a WebRoutine.
- How to use a WEB\_MAP statement to define inputs and outputs to a WebRoutine.
- How to open the Design view for a specific WebRoutine.
- How to use some of the basic editing features of the Design view.
- How to create a table.
- How to use a Weblet.
- How to compile a WAM.
- What a compile generates.

## **WAM010 - Using WEB\_MAPs**

### **Objectives**

- To demonstrate how the WEB\_MAP affects the transfer of data between two WebRoutines.

In this exercise, you will see how fields are mapped from the WebRoutine executing on the server to the page and from the page back to the WebRoutine.

First you will create a WebRoutine that will pass all of the fields as input and output. Since all of the fields will be both input to and output from the WebRoutine, the data will be preserved when it calls itself. You will make changes to the fields and test this.

You will create a second WebRoutine that will not take all of the fields as input. You will transfer between the two WebRoutines to see how some data is lost, to better illustrate how the WEB\_MAP statement works.

To achieve these objectives, you will complete the following:

[Step 1. Create a new WAM](#)

[Step 2. Add WebRoutines to the new WAM](#)

[Step 3. Compile the WAM and Open for Editing](#)

[Step 4. Add buttons to the WebRoutine](#)

[Step 5. Understand the Web Routine](#)

[Step 6. Change the Employee Number field](#)

[Step 8. Add buttons to the WebRoutines](#)

[Step 9. Understand WEB\\_MAP](#)

[Summary](#)

### **Before you Begin**

In order to complete this exercise, you should have completed the following:

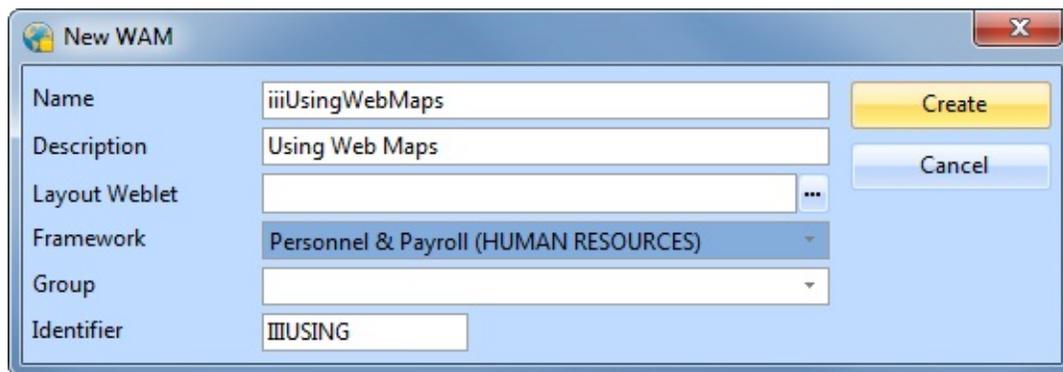
[WAM005 - Create Your First WAM](#)

## Step 1. Create a new WAM

In this step, you will create a new Web Application Module that will eventually demonstrate the use of the WEB\_MAP statement.

1. In the LANSAs Editor window, click the *New button* and choose *Web Application Module*.

The *New WAM* dialog will appear:



Name	iiiUsingWebMaps	Create
Description	Using Web Maps	
Layout Weblet		Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	
Group		
Identifier	IIIUSING	

2. In the *New WAM* dialog box:
  - a. Enter a *Name* of **iiiUsingWebMaps** (where **iii** are your initials).
  - b. Enter a *Description* of **Using WEB\_MAPs Demo**.
  - c. Leave the *Layout Weblet* field blank.
  - d. Select the *Personnel & Payroll Framework*.
  - e. Click the *Create* button to create the new WAM.
3. The LANSAs Editor will now display the WAM's RDMLX code. At this stage, it will not contain any WebRoutines.

## Step 2. Add WebRoutines to the new WAM

In this step, you will add the RDMLX code consisting of multiple WebRoutines to the newly created WAM.

1. Immediately following the BEGIN\_COM, insert the following RDMLX code to create a WebRoutine named WMdemo:

```
Webroutine Name(WMdemo) Desc('WEB_MAP WR1')
Endroutine
```

2. All fields in the WebRoutine will be both incoming and outgoing, so they will be specified FOR(\*BOTH). By default, all of the fields will be displayed as input fields. You could write the following WEB\_MAP statement for the WebRoutine:

```
WEB_MAP FOR(*BOTH) FIELDS(#EMPNO #GIVENAME #SURNAME #
```

However, a GROUP\_BY may be used in a WEB\_MAP, so you will use the following code:

```
Begin_Com Role(*EXTENDS #PRIM_WAM)
Group_By Name(#Empdata) Fields(#empno #surname #givename #address1
#postcode)
Webroutine Name(WMdemo) Desc('WEB_MAP WR1')
Web_Map For(*BOTH) Fields(#empdata (#stdrentry *hidden))
... WebRoutine
```

The STDREENTRY field will be used to control the logic within the WAM. Included in this logic will be an IF Statement that will test the value of STDREENTRY to determine if the other fields should be replaced (Refreshed) with data from the Personnel Master File.

The STDREENTRY field should not be visible on the HTML page.

The other fields will be used to demonstrate how the WEB\_MAP works.

**Note:** A WEB\_MAP statement with Keyword FOR(\*BOTH) specifies that the fields listed in the WEB\_MAP are both input to and output from the WebRoutine.

A WEB\_MAP FOR(\*INPUT) is for fields that will only be received as input to the WebRoutine.

A FOR(\*OUTPUT) is used in cases where fields will be sent out from the WebRoutine only.

A FOR(\*NONE) option will be explained in a later exercise.

In the Fields() keyword of WEB\_MAP, the fields are specified with their display mode. The display mode attribute only plays a role for fields that are FOR(\*OUTPUT) or FOR(\*BOTH).

By default, all fields are displayed on the web page as input fields. To change the way the fields display on the web page, the display mode can be set to \*INPUT, \*OUTPUT, \*HIDDEN, or \*PRIVATE.

\*HIDDEN fields are included in the page as hidden fields. That is, their values are mapped but they are not shown on the web page.

\*PRIVATE fields are not shown on the web page but are available in the XML for use in weblets such as a dropdown lists.

3. Add the code to initialize the fields in the WebRoutine. Since EMPNO is both incoming to and outgoing from the WebRoutine, its value should never be lost. If EMPNO is blank, that means it is the first time entering the WebRoutine. You will retrieve a valid employee number by simply reading the first record from the Personnel Master.

```
If Cond(#empno = *blanks)
Select Fields(#empdata) From_File(pslmst)
Leave
Endselect
Endif
```

**Note the following about this code:**

The SELECT loop with an unconditional LEAVE, will return values for the first record read. Typically you should not rely on values returned outside a SELECT loop because the returned values may be unpredictable. This technique has been used for the sake of simplicity. An alternative could be:

```
Fetch Fields(#EMPDATA) From_file(PSLMST)
```

With no key specified, the FETCH will return the first record in the file.

4. A *Read* button will be required to read data from the Personnel Master file when an Employee Number has been entered on the web page. The read will do a FETCH on the Personnel Master, using EMPNO as the key and will be triggered by a button calling the WebRoutine with a STDREENTRY value of 'R'.

```
If Cond(#stdreentry = R)
```

```
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Endif
```

Your finished WebRoutine should appear as follows:

```
Group_By Name(#Empdata) Fields(#empno #surname #givenname #address1
#postcode)
Webroutine Name(WMdemo) Desc('WEB_MAP WR1')
Web_Map For(*BOTH) Fields(#empdata (#stdrentry *hidden))
If Cond(#empno = *blanks)
Select Fields(#empdata) From_File(pslmst)
Leave
Endselect
Endif
If Cond(#stdrentry = R)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Endif

EndroutineWebRoutine
```

### Step 3. Compile the WAM and Open for Editing

In this step, you will compile the WAM.

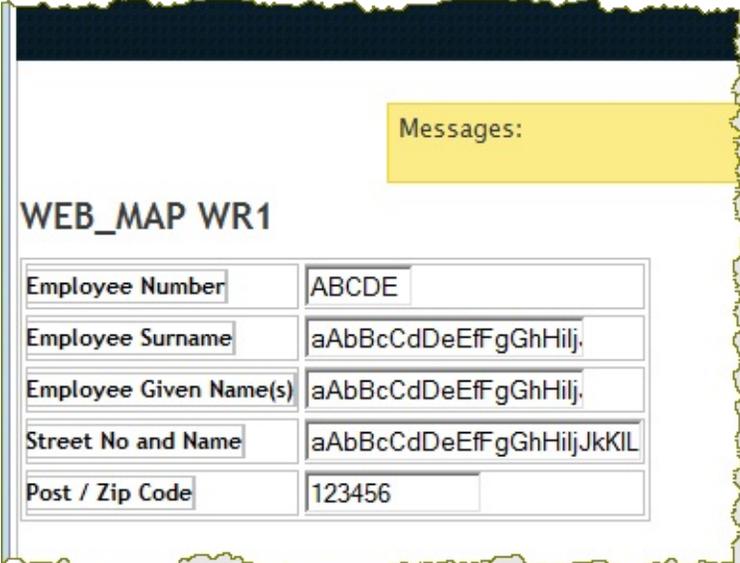
1. You should still have the WAM **iiiUsingWebMaps** (where **iii** are your initials) open in the LANSAs Editor.
2. Click the  *Compile* button on the LANSAs Editor toolbar to compile the WAM component.
3. After the compile completes successfully, open the WMdemo WebRoutine in the Design view by right clicking on the green arrow next to the WMDemo WebRoutine statement and selecting Open Design/LANSAs XHTML.

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)

  Group_By Name(#Empdata) Fields(#empno #surname #givenname #address1 #postcode)
  Webroutine Name(WMdemo) Desc('WEB_MAP WR1')
  Web_Map For(*BOTH) Fields(#Empdata (#stdre
  If Cond(#empno = *blanks)
    Select Fields(#Empdata) From_File(pslmst
    Leave
  Endselect
  Endif
  If Cond(#stdreentry = R)
    Fetch Fields(#Empdata) From_File(pslmst) With_Key(#empno)
  Endif
```



The *Design* view should appear something like the following:

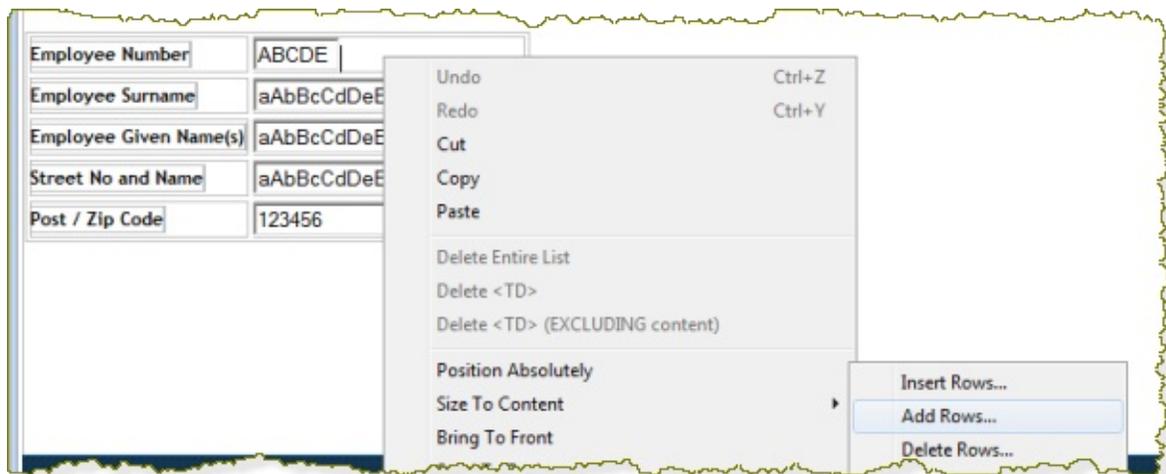


WEB_MAP WR1	
Employee Number	ABCDE
Employee Surname	aAbBcCdDeEffGhHilj.
Employee Given Name(s)	aAbBcCdDeEffGhHilj.
Street No and Name	aAbBcCdDeEffGhHiljJkKIL
Post / Zip Code	123456

## Step 4. Add buttons to the WebRoutine

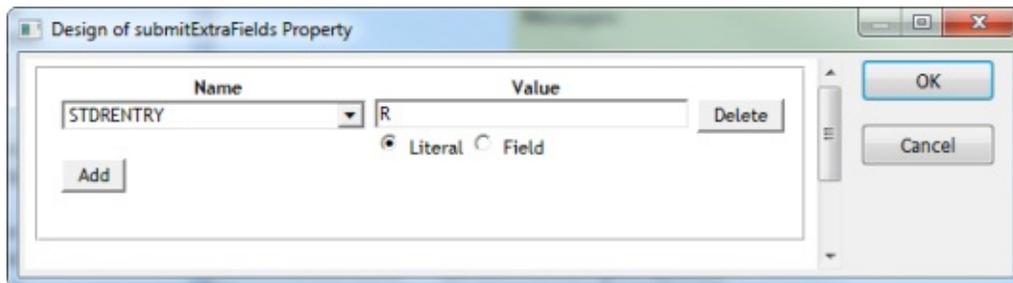
In this step, you will learn how to add a row to the generated table and then create the buttons that will invoke this WebRoutine. A reentrant WebRoutine is designed so that from the web page that is presented to the user, after the WebRoutine ends, navigation elements are provided that will allow the WebRoutine to be initiated once again. Data on the page will be used in the WebRoutine to control the logic performed by the routine. This technique is used to reduce the number of WebRoutines required for a particular application.

1. Add a new row to the table:
  - a. Right click in the table that contains the data entry fields. For example click to the right of the Employee Number field input box and then right click.
  - b. Select the *TableItems* menu item and choose the *Add Rows...* option from the pop-up menu. Click OK. A row will be added to the end of the table.



2. This step adds the *Read* button:
  - a. From the Favorites / *Weblet Templates* tab, drag and drop a *Push button* weblet into the leftmost cell of the newly added row.
  - b. Click on the empty space of the cell containing the button.
  - c. Select the *Details View*.
  - d. Set the *align* property to **left**.
3. Set the *Read* button properties:
  - a. Select the new button in the *Design View*.

- b. Set the *caption* to **Read**.
- c. Set the *on\_click\_wrname* to **WMdemo**.
- d. Click in the *Value* column for the *submitExtraFields* property and use the *Ellipsis* button to open the *Design of...* dialog. In the *Name* column select **STDREENTRY** from the dropdown list. In the *Value* column enter **R** and leave the checkbox as **Literal**. Click *OK* to close the dialog and save your changes. The button click will return the field STDREENTRY with a value of R.



4. Add the button to reload the WebRoutine:
  - a. From the *Weblet Templates* tab, drag and drop a *Push button* weblet into the rightmost cell of the newly added row.
  - b. Click on the empty space of the cell containing the button.
  - c. Select the *Details View*.
  - d. Set the *align* property to **right**.
5. Set the button's properties:
  - a. Select the new button in *Design View*.
  - b. Set the *caption* to **WMDemo**.
  - c. Set the *on\_click\_wrname* to **WMdemo**.
  - d. Click in the *Value* column for the *submitExtraFields* property and use the *Ellipsis* button to open the *Design of...* dialog. In the *Name* column select **STDREENTRY** from the dropdown list. In the *Value* column enter ' ' and leave the checkbox as **Literal**. Click *OK* to close the dialog and save your changes. The button click will return the field STDREENTRY with a blank value.

Messages:

### WEB\_MAP Demo WR1

Employee Number	ABCDE
Employee Given Name(s)	ABCDEFGHIJKLMNOPQR
Employee Surname	ABCDEFGHIJKLMNOPQR
Street No and Name	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code	123456
Read	WMDemo

6. Save the changes to WMDemo.

## Step 5. Understand the Web Routine

In this step, you will test the WebRoutine and gain an understanding of how it works.

1. WMDemo should still be open in the *Design* view, if not, open it.
2. Click the  Run button on the toolbar to run the WebRoutine in the browser.'

This will open your default Web Browser and request the URL to execute the WebRoutine **WMDemo**.

3. What happens when no logic in the WebRoutine is executed?

- a. Click on the WMDemo Button.

This will execute the WMDemo WebRoutine. Notice nothing has changed or been lost.

- b. Try changing any field except Employee Number. Click the WMDemo button again to see that the data has been preserved. The data on the web page was sent to the WMDemo WebRoutine where the WEB\_MAP FOR(\*BOTH) accepted the values as input.

The RDML logic will not be executed as EMPNO is not blank and STDREENTRY is not equal **R**. When the ENDROUTINE is encountered, the field's values are sent out to the web page as the WEB\_MAP is designated FOR(\*BOTH), that is both input and output.

4. Let us look at the effect when some of the logic in the WebRoutine is executed.

As you have changed some fields, click the *Read* button. Recall, the read logic, that does a FETCH on the Personnel Master with the Employee Number as the key.

The data on the web page should be refreshed with the values for the key EMPNO. This is because the field STDREENTRY is set to R.

The WEB\_MAP accepts the EMPNO and STDREENTRY fields along with other data from the page. The FETCH using the EMPNO field is executed and the data for the other fields on the page is set to the values from the master file PSLMST.

If there is no employee record for the value of EMPNO, the original values are returned to the web page. When the ENDROUTINE is encountered, the

field's values are sent out to the web page as the WEB\_MAP is designated FOR(\*BOTH) that is both input and output.

5. What happens if you change the Employee Number field? Or change Employee Number, as well as some other fields?
  - a. Now click the WMDemo button? The data is preserved , as it should be, as long as employee number is not blank.
  - b. Click the *Read* button. If an employee number that exists on the PSLMST was entered, the details for that employee will be displayed. Suitable employee number values are A0090, A0070 or A1001.

In both cases, when the ENDROUTINE is encountered, the field's values are sent to the web page as the WEB\_MAP is designated FOR(\*BOTH) meaning the fields are both input and output from the WebRoutine.

6. To force the WebRoutine to select the first employee again, click the WMDemo button with no employee number present.

By this point you should understand what the buttons are doing.

This exercise demonstrated how WEB\_MAPS with all fields with a FOR(\*BOTH) keyword send and accept data from the web page.

## Step 6. Change the Employee Number field

In this step, you will change Employee Number to be an output only field. This will prevent the field from being changed and change the behavior you experienced in the last step. You need to be aware that fields that are displayed as **output** are generated as **text** on the page. Since it is text and not a field, the necessary tags to pass the field to the next WebRoutine are not generated.

You must add the field EMPNO a second time as a hidden field to preserve the data.

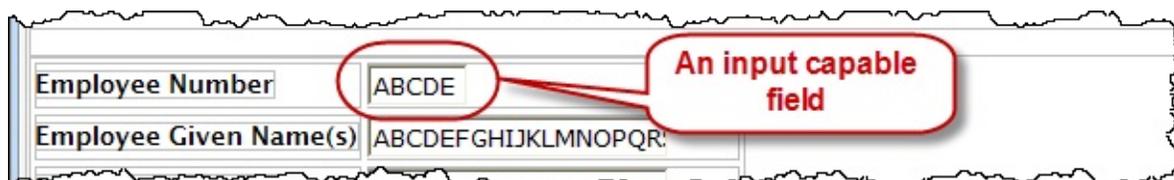
You will also see the dangers of generating XSL for ALL WebRoutines on compile, as opposed to only generating the XSL for new WebRoutines.

1. Change the GROUP\_BY to display Employee Number as output only.

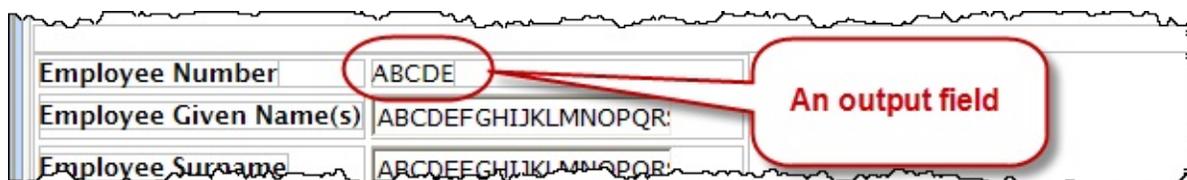
```
Group_By Name(#Empdata) Fields((#empno *output) #surname #givenname #address1 #postcode)
```

2. Click the *Compile* button in the LANSa Editor toolbar to compile the WAM component.
3. After the compile completes successfully, reopen the WMdemo WebRoutine in the Design view.
4. Notice that the Employee Number field has not changed. You could still input a value to it. This is because a compile, by default, generates the XSL for NEW WebRoutines only.

An input capable field in the *Design* view:



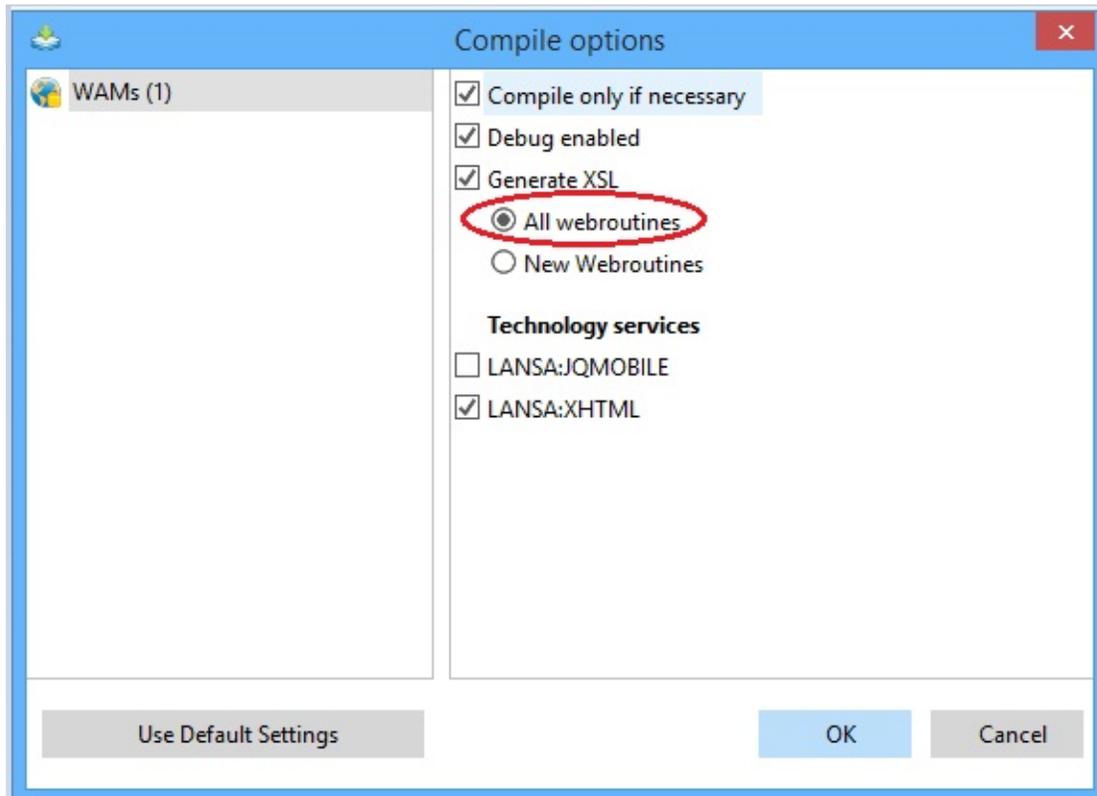
An output field in the *Design* view:



**Note:** This is the safe *Compile* option because the web page design for all existing WebRoutines will remain unchanged when a WAM is recompiled.

You may choose to re-generate the XSL for all WebRoutines by changing the default of the *WAM Compile Options* from *New WebRoutines* to *All WebRoutines*. If this is done then all XSL Editing changes will be lost for all WebRoutines in the WAM. Therefore you would only select this option if you wish to discard all your web page design changes for the WAM's WebRoutines.

The following images depicts the effect on WMDemo if this option was taken. **Please do not perform these steps.** They are only for information. You will continue the exercise at point 5.



If check *All WebRoutines* in the *Compile options* window is selected, the Design view would appear something like the following:

Messages:

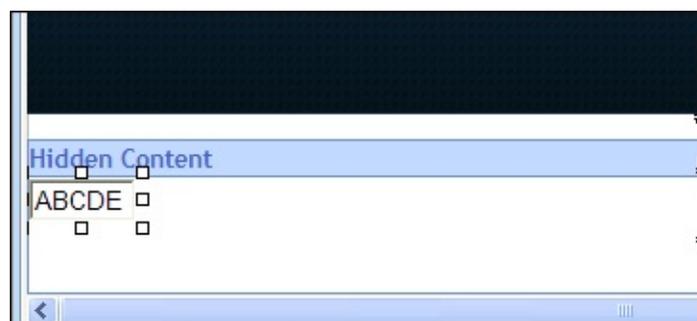
**WEB\_MAP Demo WR1**

Employee Number	ABCDE
Employee Given Name(s)	ABCDEFGHIJKLMNOPQR
Employee Surname	ABCDEFGHIJKLMNOPQR
Street No and Name	aAbBcCdDeEfGhHiIjJkKlLm
Post / Zip Code	123456

Notice Employee Number is displayed as an output field now, but the buttons added in the previous step have been lost.

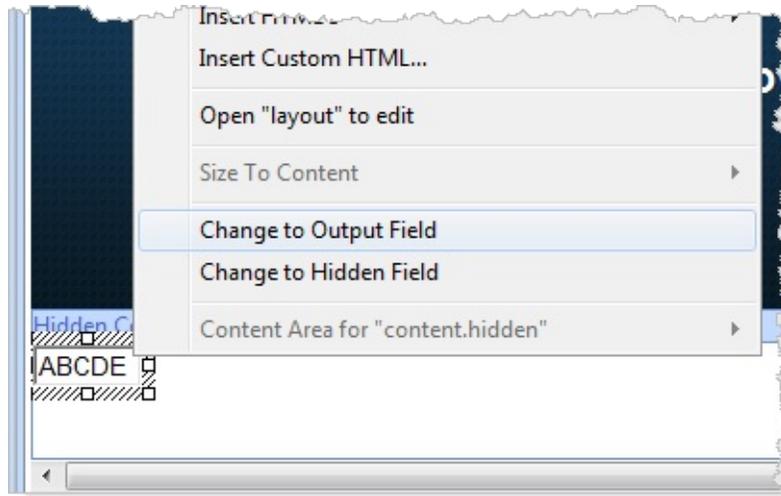
If you were to take this approach, you would have to re-add a row to the table and recreate the buttons.

5. Add EMPNO to the page again.
  - a. From the WebRoutine Output *tab*, select the EMPNO field and drag it into the top right cell of the table, to the right of the EMPNO field that is currently on the page.
  - b. Delete the caption that was added when you added the field (this caption will already be selected after dragging the field onto the page).
6. Hide the input capable instance of EMPNO
  - a. Drag the original EMPNO field (not the one you just added) into the *Hidden Content* area at the bottom of the page.



Fields in the *Hidden Content* DIV will not be displayed on the page in the browser, but will be visible in the *Design View*.

If you would like to hide the field in the *Design View* also, right click on it and select *Replace with hidden field*.



**Note:** You can change the display mode of a field at any time in the *Design* view by right clicking the field and selecting *Change to Input Field*, *Change to Output Field*, or *Change to Hidden Field*, without having to change the WEB\_MAP.

However, note that if you want to map the field into the WebRoutine, the Web\_Map must have a FOR(\*INPUT) or FOR(\*BOTH) parameter.

7. Save your changes and test the page again.

## Step 7. Add the RDMLX for the second WebRoutine

In this step, you will create the RDMLX code for a second WebRoutine to demonstrate how data is passed between WebRoutines.

1. Immediately following the ENDRoutine for WMdemo, insert the following RDML code to create a WebRoutine named WMdemo2:

```
Webroutine Name(WMdemo2) Desc('Web_Map Demo WR2')
Endroutine
```

2. This WebRoutine will require two outgoing fields (ADDRESS1 and POSTCODE) and four fields that are both incoming and outgoing (GIVENAME, SURNAME, EMPNO and STDREENTRY). EMPNO and STDREENTRY must be FOR(\*BOTH) to support the functionality of the WebRoutine. The other four fields were arbitrarily divided to show the behavior of the WEB\_MAP, but don't necessarily represent a practical use. Add the following WEB\_MAP statements to the WebRoutine:

```
Web_Map For(*BOTH) Fields((#empno *output) #givename #surname
(#stdrentry *hidden))
Web_Map For(*output) Fields(#Address1 #POSTCODE)
```

The **STDREENTRY** field will be used to determine what button was pressed, and side what logic to execute.

The **STDREENTRY** field should not be visible on the HTML page.

The **EMPNO** field should be output only, so the user cannot change it.

The other fields will be used to demonstrate how the WEB\_MAP works.

3. The logic for the *Read* button will be the same as in WMdemo, that is, it will **FETCH** the record from the Personnel Master when the **STDREENTRY** value is 'R'.

```
If Cond(#stdrentry = R)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Endif
```

Your finished WebRoutine should appear as follows:

```
Webroutine Name(WMdemo2) Desc('Web_Map Demo WR2')
Web_Map For(*BOTH) Fields((#empno *output) #givename #surname
(#stdrentry *hidden))
Web_Map For(*output) Fields(#Address1 #POSTCODE)
If Cond(#stdrentry = R)
```

```
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Endif
Endroutine
```

4. Click the  *Compile* button in the LANSAs Editor toolbar to compile the WAM component.

## Step 8. Add buttons to the WebRoutines

In this step, you will add a row to the table and then create the buttons that will reload the WebRoutine, as you did in [Step 4. Add buttons to the WebRoutine.](#)

You will also add a button to call WMdemo.

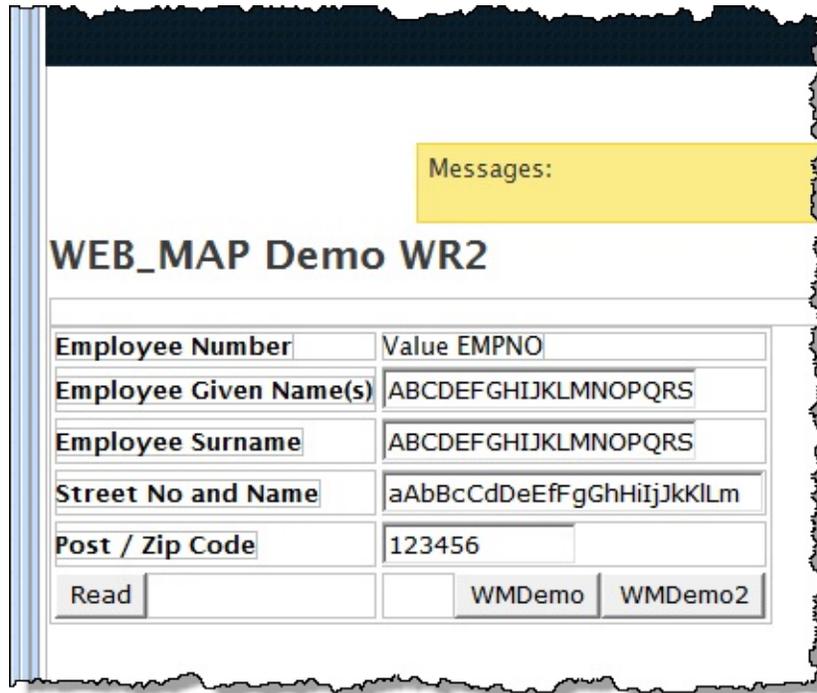
Finally, in the WMdemo web page you will add a button to call WMdemo2.

1. Open the new WebRoutine, WMdemo2 in the Design view.
2. This step adds a new row to the bottom of the table:
  - a. Right click on the table that contains the data entry fields.
  - b. Select the *Table Items* menu item and choose the *Add Rows* option from the pop-up menu.
3. Now add the *Read* button:
  - a. From the Favorites/*Weblet Templates View*, drag and drop the *Push button* weblet into the leftmost cell of the new row.
  - b. Click on the empty space of the cell containing the button.
  - c. Select the *Details View*.
  - d. Set the *align* property to **left**.
4. Set the *Read* button properties:
  - a. Select the new button in *Design View*.
  - b. Set the *caption* property to **Read**.
  - c. Set the *on\_click\_wrname* property to **WMdemo2**.
  - d. Click the Ellipsis button for the *submitExtraFields* property and set the returned field *Name* to **STDREENTRY** and the *Value* to **R** to indicate that the users wants to refresh.
  - e. Click *OK* to confirm the change.
5. This step adds the button to reload the WebRoutine:
  - a. From the Favorites/*Weblet Templates View*, drag and drop the *Push button* weblet into the rightmost cell of the newly added row.
  - b. Click on the empty space of the cell containing the button.
  - c. Select the *Details View*.
  - d. Set the *align* property to **right**.

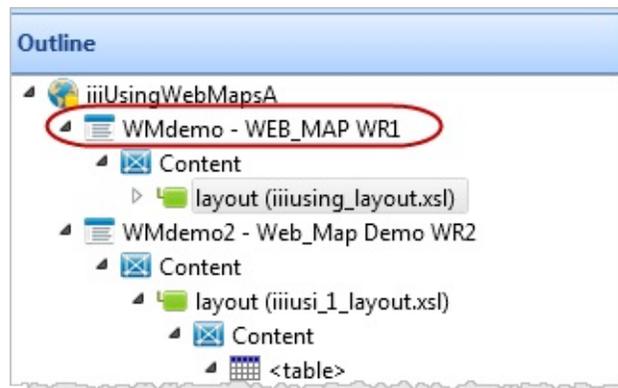
6. Set the button's properties:
  - a. Select the new button in *Design View*.
  - b. Set the *caption* property to WMDemo2.
  - c. Set the *on\_click\_wrname* property to **WMdemo2**.
  - d. Click in the *Value* column for the *submitExtraFields* property and use the *Ellipsis* button to open the *Design of...* dialog.
  - e. In the *Name* column select **STDREENTRY** from the dropdown list. In the *Value* column enter ' ' and leave the checkbox as **Literal**.
  - f. Click *OK* to close the dialog and save your changes. The button click will return the field STDREENTRY with a blank value.
7. Add a hidden copy of the field EMPNO to the page:
  - a. From the WebRoutine Output View, select the EMPNO field and drag it into the top right cell of the table.
  - b. Delete the caption that was added when you added the field.
  - c. Change the new EMPNO field to an Input Field.
  - d. Drag the new EMPNO field into the *Hidden Content* area at the bottom of the web page.
  - e. If you would like to hide the field in the *Design View* also, right click on it and select *Change to hidden field*.

**Note:** The hidden EMPNO field is required so that a value will be mapped into the WebRoutine WMDemo2, when the WMDemo2 button is clicked.
8. Add the button to transfer to WMdemo. From the Favorites/*Weblet Templates View*, drag and drop the *Push button* weblet into the rightmost cell of the newly added row.
9. Set the button's properties:
  - a. Select the new button in *Design View*.
  - b. Set the *caption* property to **WMDemo**.
  - c. Set the *on\_click\_wrname* property to **WMdemo**.
  - d. Click in the *Value* column for the *submitExtraFields* property and use the *Ellipsis* button to open the *Design of...* dialog. In the *Name* column select **STDREENTRY** from the dropdown list. In the *Value* column enter ' ' and leave the checkbox as **Literal**. Click *OK* to close the dialog and save your

changes. The button click will return the field STDRENTRY with a blank value



- 10. Save your changes to WebRoutine WMdemo2.
- 11. Open WebRoutine WMdemo in the Design view. To do this, select the *Outline* tab and double click on the WMdemo entry:



- 12. Add a new button in WMdemo, to invoke the WebRoutine WMdemo2. From the Favorites/*Weblet Template View*, drag and drop the *Push button* weblet into the rightmost cell of the bottom row.
- 13. Set the button's properties:
  - a. Select the new button in *Design View*.

- b. Set the *caption* property to **WMDemo2**.
- c. Set the *on\_click\_wrname* property to **WMdemo2**.
- d. Click in the *Value* column for the *submitExtraFields* property and use the *Ellipsis* button to open the *Design of...* dialog.
- e. In the *Name* column select **STDREENTRY** from the dropdown list. In the *Value* column enter ' ' and leave the checkbox as **Literal**.
- f. Click *OK* to close the dialog and save your changes. The button click will return the field STDREENTRY with a blank value.

Messages:

**WEB\_MAP Demo WR1**

Value EMPNO

<b>Employee Number</b>	
<b>Employee Given Name(s)</b>	ABCDEFGHIJKLMNOPS
<b>Employee Surname</b>	ABCDEFGHIJKLMNOPS
<b>Street No and Name</b>	aAbBcCdDeEffgGhHiIjKlLm
<b>Post / Zip Code</b>	123456
<b>Read</b>	<input type="checkbox"/> WMDemo2 <input type="checkbox"/> WMDemo

14. Save your changes to WMdemo.

## Step 9. Understand WEB\_MAP

In this step, you will test out the WebRoutines that you have just created and see the WEB\_MAP in action.

1. Open WMDemo in the Design view.
2. Use the Execute button in the toolbar to run the WebRoutine in the *web browser*.
3. Click the WMDemo2 button and observe that some of the data in these fields is not transferred to the WMDemo2 WebRoutine.

This is because ADDRESS1 and POSTCODE are output only, so although they are sent to WMDemo2 by clicking the WMDemo2 button on the WMDemo web page and are shown on the web page for WMDemo2, they are not accepted by the WMDemo2 WebRoutine as input, therefore the values of the fields are lost when transferring to the WebRoutine WMDemo2.

4. Click the *Read* button to populate all of the fields again. Now click the WMDemo button. See that all of the data is preserved. All of the fields are output from WMDemo2 and they are all accepted as input to WMDemo, so data will not be lost.
5. The employee number is always retained in its output field because the web page for WMDemo and WMDemo2 WebRoutines contain a hidden copy of employee number, which is mapped into the WebRoutine.

**Note:** In web applications, it is often necessary to display a key field value for output and have a hidden copy of the field on the page, in order to pass a value into the next WebRoutine.

## Summary

### Important Observations

- The WEB\_MAP statement allows you to specify which incoming and outgoing fields the WebRoutine maps between the web page and the RDMLX code. The WEB\_MAP statement's FOR() selector specifies whether the fields are mapped as incoming (\*INPUT), outgoing (\*OUTPUT), or both (\*BOTH).
- Only fields in a WEB\_MAP statement with a FOR(\*OUTPUT) or FOR(\*BOTH) can be placed on the web page.
- The Fields() attributes specify the display mode of the field on the page. Acceptable field attributes are \*INPUT, \*OUTPUT and \*HIDDEN. If unspecified, the default is \*INPUT. These attributes determine whether the field accepts input, as an input text box, only displays an output value, or is hidden on the page.
- Hidden fields may be needed if fields that are shown as output values on the web page, but their value must be posted to the web server. This is often the case for key fields when the WebRoutine is performing an update.
- The order in which fields appear when generated is the same as their sequence in the WEB\_MAPs, although when data is mapped to and from the WEB\_MAPs they are mapped by field names and not position.
- A WebRoutine may have multiple WEB\_MAPs.
- If the output of one WebRoutine does not match the input of the WebRoutine that it is calling, no error will be generated. The data will simply not be passed into the WebRoutine that was called.

### Tips & Techniques

- In addition to specifying WEB\_MAPs inside WebRoutine blocks, you are allowed to declare WEB\_MAPs inside a BEGIN\_COM block of a WAM. This technique allows you to map fields and lists into every WebRoutine in your WAM without having to explicitly define WEB\_MAPs in each WebRoutine.
- You can change the WEB\_MAP after the first compile, but you must update the web page for the fields affected. e.g. if a field's display attribute was changed in the WEB\_MAP from \*input to \*output, the web

page could be corrected either by changing the field on the page to "Output Field" or delete the field and drag and drop it onto the page from the WebRoutine Output tab.

- You can modify the display mode of a field in the Design view without having to change the WEB\_MAP by right clicking the field and changing to the appropriate display mode. However, note that if you change a field to be \*INPUT in the *Design* view, then it must be mapped into the WebRoutine using For(\*input) or For(\*both) in order for the value to be processed.
- Similarly if you change a field to be \*OUTPUT in the *Design* view, and it is mapped into the WebRoutine as For(\*input) the field value will never be received by the WebRoutine.
- For consistency, we recommend you control field input and output attributes via their WEB\_MAP statements, rather than by tweaking the *Design* view.

## **What I Should Know**

- How the WEB\_MAP works.

## **WAM015 - Working Lists**

### **Objectives**

- To demonstrate how to use and change Working Lists on the page.

In this exercise, you will create a WAM with a single WebRoutine that will populate a working list. This WebRoutine will have no other function, but will teach you how to use a working list on a page and how to condition the field's display modes.

To achieve this Objective, you will complete the following:

[Step 1. Create a new WAM](#)

[Step 2. Add RDMLX code to the new WAM](#)

[Step 3. See how the working list is displayed](#)

[Step 4. Change the display mode of fields in the list](#)

[Step 5. Use the generate XSL for all WebRoutines option](#)

[Step 6. Modify the list in the Design view](#)

[Summary](#)

### **Before you Begin**

In order to complete this exercise, you should have completed the following:

- [WAM005 - Create Your First WAM](#)
- [WAM010 - Using WEB\\_MAPs](#)

## Step 1. Create a new WAM

In this step, you will create a new Web Application Module that you will use to become familiar with working lists.

1. In the LANSAs Editor window, click the *New button* choose *Web Application Module*.
2. In the *New WAM* dialog box:
  - a. Enter a *Name* of **iiiWorkingLists** (where **iii** are your initials).
  - b. Enter a *Description* of **Working List Demo**.
  - c. Leave the *Layout Weblet* value blank.
  - c. Click the *Create* button to create the new WAM.
3. The LANSAs Editor will now display the WAM's RDMLX code. At this stage, it will not contain any WebRoutines.

## Step 2. Add RDMLX code to the new WAM

In this step, you will add the RDMLX code to the newly created WAM.

1. Create a WebRoutine named **ListMain** with a description of **List Demonstration**.
2. Define a list that will contain information from the Personnel Master file.

```
Def_List Name(#emplist) Fields(#empno #givenname #surname #address1  
#phonehme) Type(*working) Entrys(*max)
```

**Note:** In RDMLX programs you should usually define lists using Entrys(\*max). As well as having an upper limit which is only limited by the platform, lists defined in this way are dynamic and only occupy the memory required for their current number of entries.

3. Add code to the WebRoutine to populate the list with data from PSLMST.

```
Clr_List Named(#EMPLIST)  
Select Fields(#emplist) From_File(pslmst)  
Add_Entry To_List(#emplist)  
Endselect
```

4. Create a WEB\_MAP and add list EMPLIST to it, so the list can be displayed on the page.

Your finished WebRoutine should appear as follows:

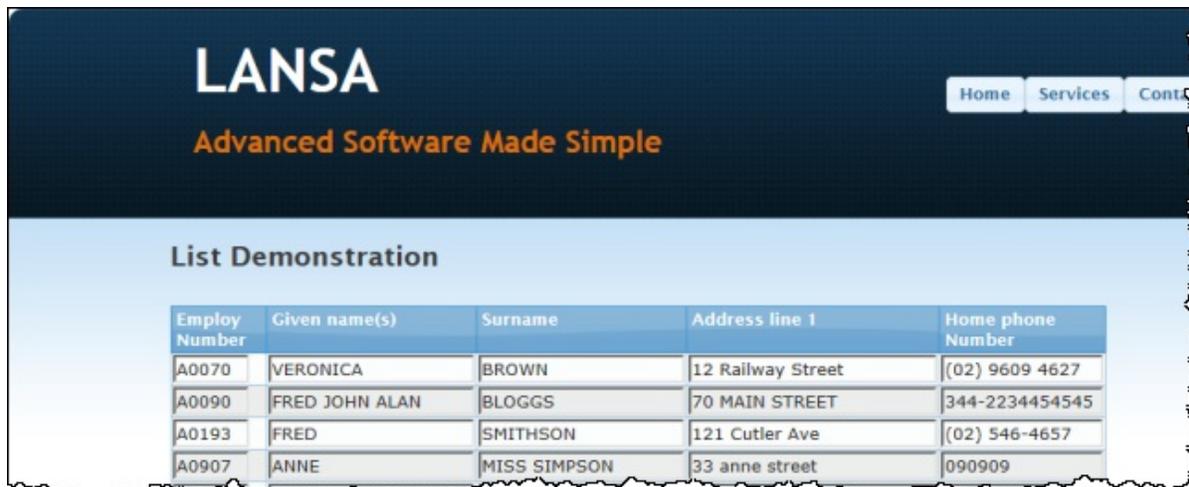
```
Webroutine Name(ListMain) Desc('List Demonstration')  
Web_Map For(*both) Fields(#emplist)  
Def_List Name(#emplist) Fields(#empno #givenname #surname #address1  
#phonehme) Type(*working) Entrys(*max)  
Clr_List Named(#EMPLIST)  
Select Fields(#emplist) From_File(pslmst)  
Add_Entry To_List(#emplist)  
Endselect  
Endroutine
```

5. Compile the WAM. When you compile a WAM, it is always saved first.

### Step 3. See how the working list is displayed

In this step, you will see how the list is displayed on the page.

1. Open the WebRoutine ListMain in the Design view.
2. In the Editor, click the *Execute* button on the toolbar to run the WebRoutine in the web browser.



The screenshot shows a web application interface. At the top, there is a dark blue header with the "LANSA" logo in white and the tagline "Advanced Software Made Simple" in orange. To the right of the logo are three navigation buttons: "Home", "Services", and "Contact". Below the header is a light blue section titled "List Demonstration". In the center of this section is a table with five columns: "Employ Number", "Given name(s)", "Surname", "Address line 1", and "Home phone Number". The table contains four rows of data, each representing an employee record.

Employ Number	Given name(s)	Surname	Address line 1	Home phone Number
A0070	VERONICA	BROWN	12 Railway Street	(02) 9609 4627
A0090	FRED JOHN ALAN	BLOGGS	70 MAIN STREET	344-2234454545
A0193	FRED	SMITHSON	121 Cutler Ave	(02) 546-4657
A0907	ANNE	MISS SIMPSON	33 anne street	090909

Notice all of the fields in the list are input fields. When you define the fields in the DEF\_LIST statement, you can specify the display mode just like in the WEB\_MAP statement.

## Step 4. Change the display mode of fields in the list

In this step, you will change the display mode of the fields in the list.

1. Change the field definitions in the DEF\_LIST command to output and make PHONEHME hidden.

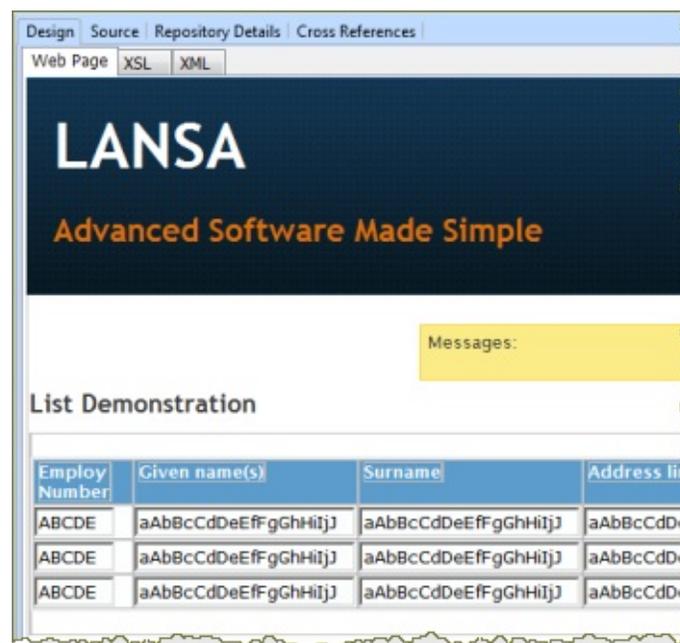
```
DEF_LIST NAME(#EMPLST) FIELDS((#EMPNO *OUTPUT) (#GIVENAM
```

**Hint:** You could use the *F4 Command Assistant*, and expand the *Fields* parameter, enabling you to enter an *\*output* attribute to each field.

If the *Assistant* is initially shown with the bottom tabs, you can use the left hand dotted bar, to drag, float and resize it. When you close the *Assistant*, the size and position is remembered.

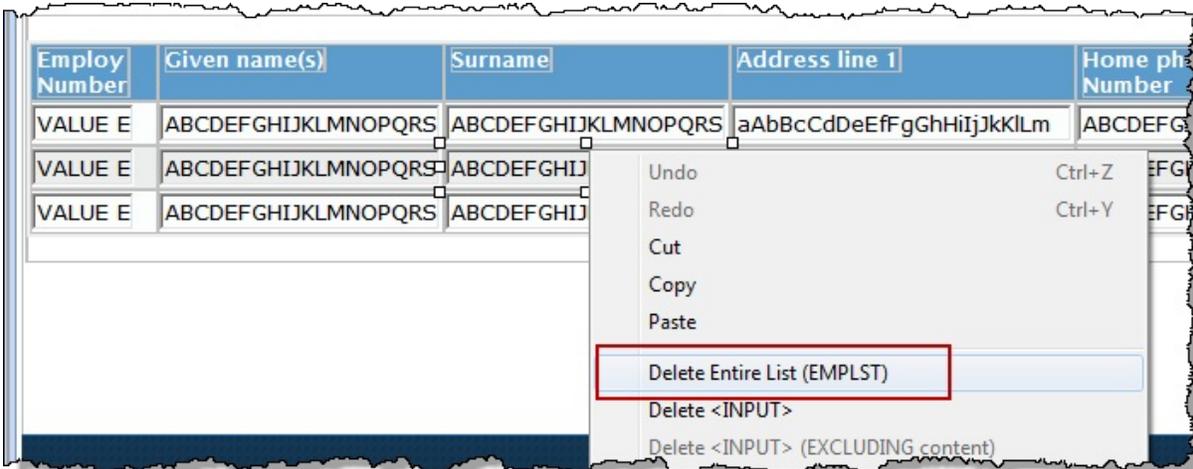
When you click in each field it will be selected. Move the cursor right, add a space and type *\*out*. The value *\*output* and *\*out* are synonymous.

2. Compile the WAM and open WebRoutine ListMain in the *Design* view.

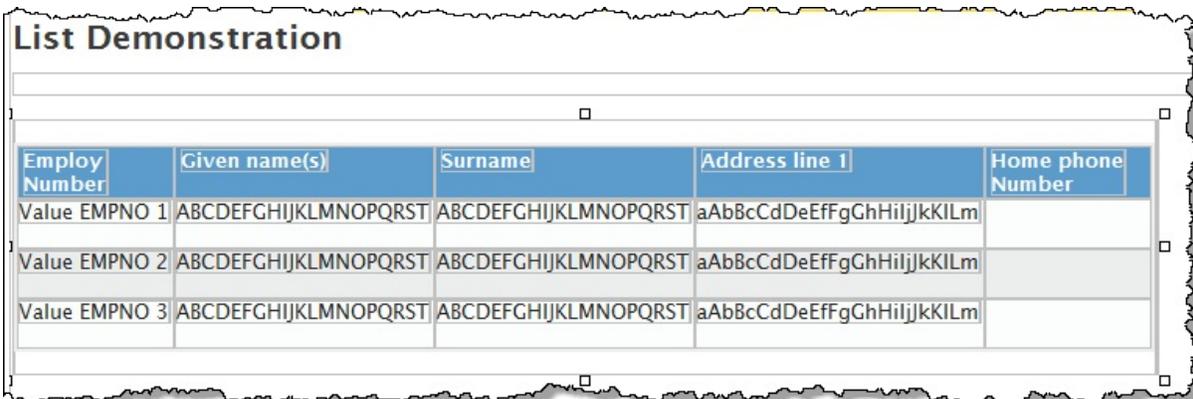


Notice all the fields are still showing as input fields. For existing WebRoutines, changes made to the WEB\_MAP statements, after the first compile do not affect the page design.

3. To update the list definition, remove the list from the page and add it back:
  - a. Right click on an empty space in the list and select *Delete Entire List*.



- b. Open the WebRoutine Output view and drag the list back onto the page. Notice that the fields in the list are now correctly shown as output.
- c. Save your changes.



4. Click the *Execute* button on the toolbar to run the WebRoutine in the browser and see how the page will look.

**Notice:** Home phone number column is visible in the *Design* view, but is hidden when you run the WebRoutine in the browser.

## Step 5. Use the generate XSL for all WebRoutines option

In this step, you change the WEB\_MAP again, but instead of removing the list from the page and adding it again, you will generate the XSL for all WebRoutines.

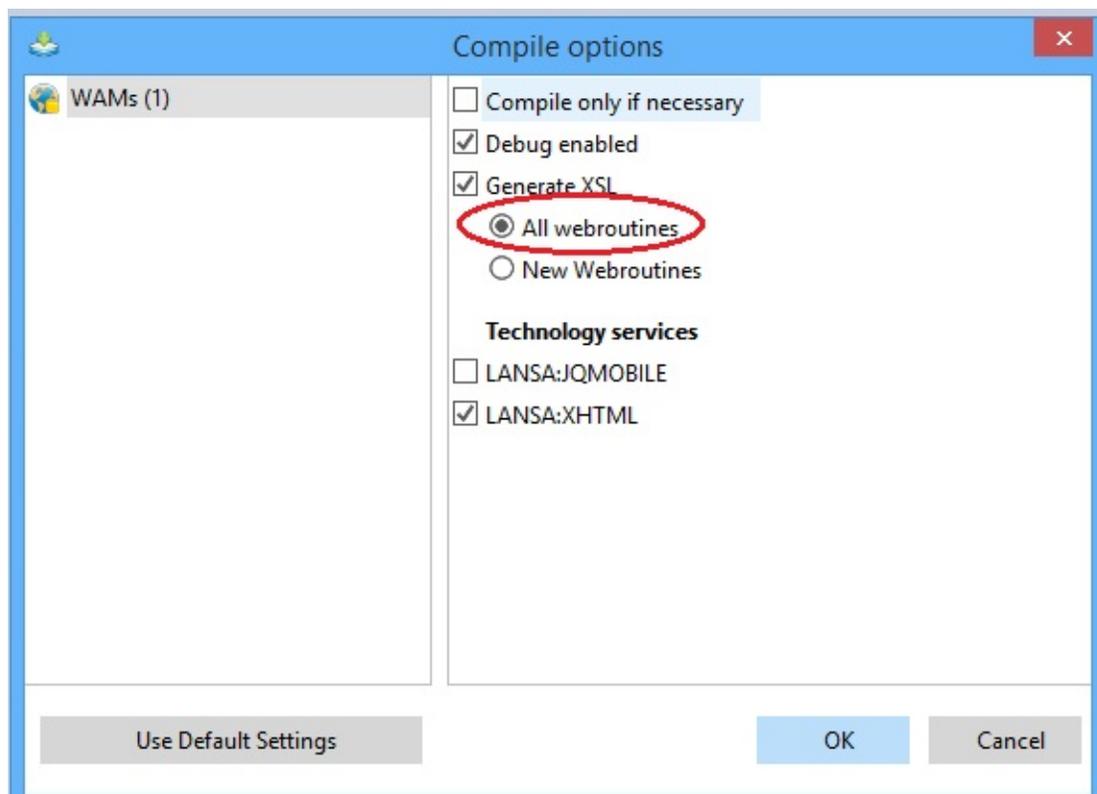
1. Change the field definitions in the DEF\_LIST command to input, or do not specify a display mode so they default to input.

```
Def_List Name(#emplist) Fields(#empno #givenname #surname #address1  
#phonehme) Type(*working) Entrys(*max)
```

2. Save your changes.
3. Open the *Compile options* dialog by clicking the *Menu* button on *Compile* section of the *Home* ribbon.



4. Make sure *Generate XSL* is checked and check *All WebRoutines*.



5. Click *OK*.
6. Open ListMain in the Design view again. Notice the list fields are all input capable again.

**Remember:** When you generate the XSL for All WebRoutines, changes you have made to **ANY** WebRoutines in the WAM will be lost.

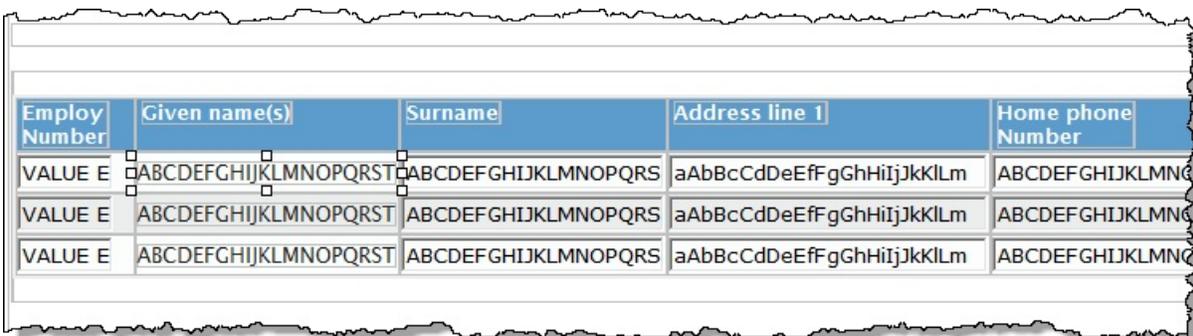
## Step 6. Modify the list in the Design view

In this step, you will make changes to the list inside the Design view.

Change the field definitions in the Design view by right clicking on the field in the list and selecting one of *Change to Input Field*, *Change to Output Field*, or *Change to Hidden Field*. When you make this change in the *Design* view, the field attribute is not modified in the RDMLX.

1. Change GIVENAME to an output field, by right clicking on the field and selecting *Change to Output Field*.

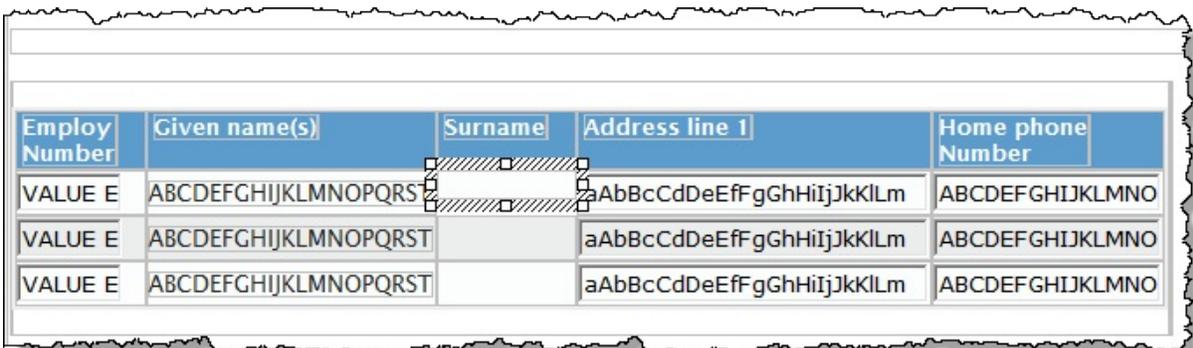
After making the change, the list in the Design view will appear something like the following:



Employ Number	Given name(s)	Surname	Address line 1	Home phone Number
VALUE E	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO
VALUE E	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO
VALUE E	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO

2. Change SURNAME to a hidden field, by right clicking on the field and selecting *Change to Hidden Field*.

After making the change, the list in the Design view will appear something like the following:



Employ Number	Given name(s)	Surname	Address line 1	Home phone Number
VALUE E	ABCDEFGHIJKLMNQRST		aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO
VALUE E	ABCDEFGHIJKLMNQRST		aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO
VALUE E	ABCDEFGHIJKLMNQRST		aAbBcCdDeEfFgGhHiIjJkKlLm	ABCDEFGHIJKLMNO

3. Save your changes to WebRoutine LISTMAIN.
4. Now change Surname to an output field.

Since SURNAME is hidden, you will not immediately be able to select it in

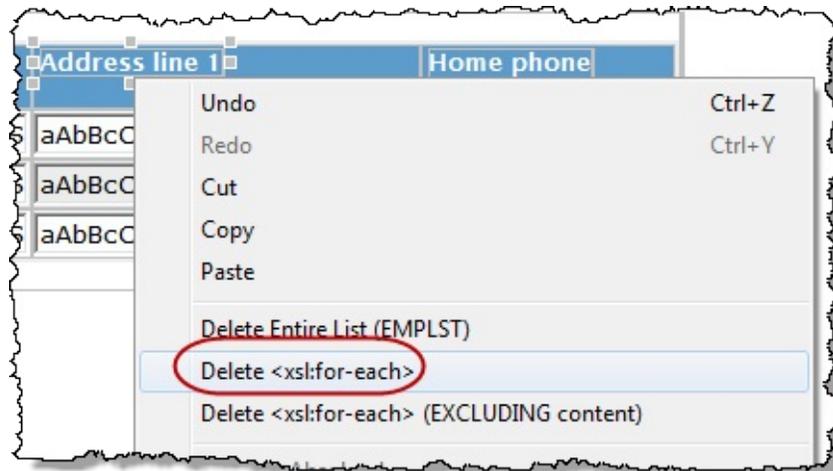
the *Design View*. To change it, take the following steps:

- a. Select the *Outline* tab.
- b. If necessary, expand the tree view so that field SURNAME is shown. Select SURNAME.
- c. Select the Details tab. This will show the properties for SURNAME.
- d. On the *Details* tab, change the Type property to text. Select this value from the dropdown list.



The Surname column will be shown as input. You may need to re-open the WebRoutine in the *Design* view to refresh the list's appearance.

5. Change the caption for the ADDRESS1 column.
  - a. Select the caption text *Address Line 1*. You should see grips around the selection.
  - b. Delete the Address Line 1 heading text using the Delete key
  - c. Alternatively you could use the right mouse context menu as shown in the picture following. This shows you are deleting the `<xsl:for-each>` that outputs this HTML element.



- d. Type **Street Address** where the old caption used to be.
6. Set the vertical alignment for the cell:
  - a. Place the cursor in the cell where you entered the "Street Address" caption.
  - b. Open the *Details View*.
  - c. Set the *vAlign* property to **bottom**.
7. Save your changes. Execute the WAM in the browser to test your changes.

## Summary

### Important Observations

- Hidden list columns are visible in *Design View*, but are not shown when run in the browser.
- When modifying a column in a list on a page, the resulting modifications are applied to all rows in this column.

### Tips & Techniques

- Lists can also be placed on the page by dragging and dropping a list from the WebRoutine Output view.

### What I Should Know

- How lists are rendered in the web page.
- How to customize the columns in a lists.

## **WAM020 - WAM Navigation**

### **Objectives**

- To demonstrate navigation between WebRoutines within the RDMLX code.

In this exercise, you will create a WAM that will look up an employee's information based on an employee number. This WebRoutine will use re-entrant programming.

You will then create two more WebRoutines that will do the same lookup, but will be executed differently from the main WebRoutine. This will show you different ways to navigate through WebRoutines within the RDML, as well as demonstrate the fact that there are many different ways to accomplish the same task.

To achieve this Objective, you will complete the following:

[Step 1. Create a new WAM](#)

[Step 2. Add RDMLX code to the new WAM](#)

[Step 3. Add Buttons and the Dropdown list to the WebRoutine](#)

[Step 4. Test and Understand the WebRoutine](#)

[Step 5. Add Weblet to a List](#)

[Summary](#)

### **Before you Begin**

In order to complete this exercise, you should have completed the following:

- [WAM005 - Create Your First WAM](#)
- [WAM010 - Using WEB\\_MAPs](#)
- [WAM015 - Working Lists](#)

## Step 1. Create a new WAM

In this step, you will create a new Web Application Module that you will use to learn the CALL and TRANSFER statements.

1. In the LANSAs Editor window, click the *New button* and choose *Web Application Module*.
2. In the *New WAM* dialog box:
  - a. Enter a *Name* of **iiiNavigation** (where **iii** are your initials).
  - b. Enter a *Description* of **Navigation Demo**.
  - c. Leave *Layout Weblet* blank.
  - d. Click the *Create* button to create the new WAM.
3. The LANSAs Editor will now display the WAM's RDMLX code. At this stage, it will not contain any WebRoutines.

## Step 2. Add RDMLX code to the new WAM

In this step, you will add RDMLX code, defining three WebRoutines.

1. Immediately following the `BEGIN_COM`, insert the following RDML code to create three new WebRoutines named `NavMain`, `NavCall` and `NavTrans`:

```
Webroutine Name(NavMain) Desc('Navigation Home')
```

```
Endroutine
```

```
Webroutine Name(NavCall) Desc('Navigation CALL')
```

```
Endroutine
```

```
Webroutine Name(NavTrans) Desc('Navigation TRANSFER')
```

```
EndroutineWebRoutineWebRoutine
```

2. Define a List containing `EMPNO` in WebRoutine `NavMain`. This list will hold all of the valid employee numbers from the personnel master. You will use this list to populate a dropdown list, so you will only be able to search for an employee that exists. Add the following code in the WebRoutine:

```
Def_List Name(#EMPLST) Fields(#EMPNO) Type(*WORKING)
```

```
Entrys(*MAX)
```

```
Clr_List Named(#EMPLST)
```

```
Select Fields(#EMPNO) From_File(PSLMST)
```

```
Add_Entry To_List(#EMPLST)
```

```
Endselect
```

3. Add a `GROUP_BY` immediately following the `BEGIN_COM`. This will be used in each WebRoutine's `WEB_MAP` and `FETCH` commands and to reset field values to default values.

```
Group_By Name(#empdata) Fields(#surname #givenname #address1 #address2  
#address3 #postcode #phonehme #phonebus #deptment #section #salary  
#startdte #termdate)
```

**Hint:** Use the *F4 Command Assistant* to define the `Group_By`. Use the *Fields in File* tab to select fields from file `PSLMST`.

4. All fields in the WebRoutine `NavMain` will be both incoming and outgoing, so they will be specified `FOR(*BOTH)`. All of the fields from `PSLMST` will be used.

By default, all the fields will be displayed as input fields.

The `EMPLST` list should be mapped with a `*PRIVATE` attribute, because it

will be used to populate the combo box weblet for EMPNO.

Add the following WEB\_MAP statement to the WebRoutine NavMain:

```
WEB_MAP FOR(*BOTH) FIELDS(#EMPNO #EMPDATA (#EMPLST *PRI
```

Fields and lists with a \*PRIVATE attribute will not be shown on the page and will not be mapped back into the WebRoutine.

The fields defined in the Group\_by will be used to output the data that was retrieved from the file.

5. The field STDREENTRY will be used to determine which logic to execute. It should be mapped as a hidden field into and out of each WebRoutine.

Immediately below the BEGIN\_COM add the following WEB\_MAP. This is a global WEB\_MAP which applies to all WebRoutines

```
Web_Map For(*both) Fields((#stdreentry *hidden))
```

6. In WebRoutine NavMain set up a **CASE loop** using the field **STDREENTRY** which will determine how to perform the search. Add this code **before** the CLR\_LIST and SELECT logic you added earlier.

```
Case Of_Field(#STDREENTRY)
When Value_Is('= A')
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
Message Msgtxt("Logic executed within this WebRoutine.")
When Value_Is('= B')
Transfer Toroutine(NavTrans)
Message Msgtxt("This message will not be displayed.")
When Value_Is('= C')
Call Webroutine(NavCall)
Message Msgtxt("Back from the CALL.")
When Value_Is('= D')
Message Msgtxt("Back from the TRANSFER.")
When Value_Is('= E')
#EMPDATA := *DEFAULT
EndcaseWebRoutine
```

7. Extend your logic for handling the list of employees, EMPLIST. The list needs to be built every time WebRoutine NavMain executes, but also needs to preserve the current value of employee number, EMPNO.

- a. Define a work field EMPNOW based on EMPNO
- b. Save current value of EMPNO in EMPNOW
- c. After the list is built restore EMPNO to EMPNOW, provided EMPNOW is not blank.

Your code should look like the following:

```
Define Field(#empnow) Reffld(#empno)
#empnow := #empno
Clr_List Named(#EMPLST)
Select Fields(#EMPNO) From_File(PSLMST)
Add_Entry To_List(#EMPLST)
Endselect
If (#empnow *NE *blanks)
#empno := #empnow

Endif
```

Your WAM at this stage should appear as follows:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)
Web_Map For(*both) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#surname #givenname #address1 #address2
#address3 #postcode #phonehme #phonebus #deptment #section #salary
#startdte #termdate)
Webroutine Name(NavMain) Desc('Navigation Home')
Web_Map For(*both) Fields(#empdata (#emplst *private))
Def_List Name(#EMPLST) Fields(#EMPNO) Type(*WORKING)
Entrys(*MAX)
Case Of_Field(#STDREENTRY)
When Value_Is('= A')
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
Message Msgtxt("Logic executed within this WebRoutine.")
When Value_Is('= B')
Transfer Toroutine(NavTrans)
Message Msgtxt("This message will not be displayed.")
When Value_Is('= C')
Call Webroutine(NavCall)
```

```

Message Msgtxt("Back from the CALL.")
When Value_Is('= D')
Message Msgtxt("Back from the TRANSFER.")
When Value_Is('= E')
#EMPDATA := *DEFAULT
Endcase
*
Define Field(#empnow) Reffld(#empno)
#empnow := #empno
Clr_List Named(#EMPLST)
Select Fields(#EMPNO) From_File(PSLMST)
Add_Entry To_List(#EMPLST)
Endselect
If (#empnow *NE *blanks)
#empno := #empnow
Endif
Endroutine
Webroutine Name(NavCall) Desc('Navigation CALL')
Endroutine
Webroutine Name(NavTrans) Desc('Navigation TRANSFER')
Endroutine

End_ComWebRoutineWebRoutineWebRoutineWebRoutine

```

**Note:** It is necessary to rebuild the list of employees (EMPLST) every time the NavMain WebRoutine is executed in order to populate the dropdown list of employee numbers. There are better ways to handle this, which will be covered in later exercises.

10. Create the WEB\_MAP for NavCall.

The WEB\_MAP will be very similar to the one in NavMain:

- However EMPNO is the only field that needs to be taken as input.
- All the employee fields need to be output from the WebRoutine. Use the GROUP\_BY to map the employee fields.
- Note that you have a global WEB\_MAP for the field STDREENTRY which applies to every WebRoutine.
- The field's display modes do not matter since this WebRoutine will not actually display anything.

Add the following WEB\_MAP statements to the WebRoutine:

```
Web_Map For(*BOTH) Fields(#EMPNO)
Web_Map For(*OUTPUT) Fields(#EMPDATA)
```

11. Add the FETCH to NavCall.

```
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
```

Your finished NavCall WebRoutine should appear as follows:

```
Webroutine Name(NavCall) Desc('Navigation CALL')
Web_Map For(*BOTH) Fields(#EMPNO)
Web_Map For(*OUTPUT) Fields(#EMPDATA)
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
Endroutine
```

12. Create the WEB\_MAP for WebRoutine NavTrans. These WEB\_MAPs will be identical to NavCall.

```
Web_Map For(*BOTH) Fields(#EMPNO)
Web_Map For(*OUTPUT) Fields(#EMPDATA)
```

13. Add the FETCH, set STDREENTRY to 'D' and TRANSFER back to NavMain.

```
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
#STDREENTRY := 'D'
Transfer Toroutine(NavMain)
```

Your finished NavTrans WebRoutine should appear as follows:

```
Webroutine Name(NavTrans) Desc('Navigation TRANSFER')
Web_Map For(*BOTH) Fields(#EMPNO)
Web_Map For(*OUTPUT) Fields(#EMPDATA)
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO)
#STDREENTRY := 'D'
Transfer Toroutine(NavMain)
Endroutine
```

14. Save and compile the WAM.

### Step 3. Add Buttons and the Dropdown list to the WebRoutine

In this step, you will add a row to the fields table and then add the buttons that will reload the WebRoutine. The NavMain WebRoutine is reentrant, that is, it calls itself.

1. Open NavMain in the Design view.
2. Delete the "Employee Number" caption by highlighting it and pressing *Delete* on the keyboard.
3. Move the EMPNO field to the top left cell, where you just deleted the caption.
  - a. Click on the EMPNO field, it will be selected with "grips" around it.
  - b. Drag the field to the top leftmost cell (where the "Employee Number" caption was deleted from in the previous step).
4. Add and configure the combo box weblet:
  - a. In the *Design View*, drag a *Combo Box* weblet from the *Favorites/Weblet Templates* tab and drop it on top of the EMPNO field.
  - b. Set up the combo box properties as:

<b>Property</b>	<b>Value</b>
listname	<b>EMPLST</b>
codefield	<b>EMPNO</b>
captionfield	<b>EMPNO</b>

- c. Notice the *name* and *value* properties are already set. By dragging the weblet onto the field, the weblet will inherit the name and value of the field.
5. Change the *colspan* property of the top leftmost cell to 2.

Your design should now look like the following:

The screenshot shows a web form titled "Navigation Home" with a yellow header bar. Below the title is a dropdown menu showing "ABCDE". The form consists of several input fields, each with a label and a placeholder value:

Employee Surname	ABCDEF GHIJKLMNOPQRS
Employee Given Name(s)	ABCDEF GHIJKLMNOPQRS
Street No and Name	aAbBcCdDeEfFgGhHiIjJkKlLm
Suburb or Town	aAbBcCdDeEfFgGhHiIjJkKlLm
State and Country	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code	123456

6. Fields Department code and Section may have combo box field visualizations defined in the Repository. If necessary use the context menu to change these to *Replace with input field*.
7. Add four buttons to the top row of the table (to the right of the combo box):
  - a. From the *Weblet Templates* tab, drag and drop four *Push button* weblets into the top row. Insert a space between each of the weblets.
  - b. Set the push button *caption* properties to **Search**, **Transfer Search**, **Call Search** and **Clear**. Adjust the width of each button if necessary.
  - c. Set the *on\_click\_wrname* property to **NavMain** for each button.
  - d. Select each push button, and click the Ellipsis button for the *submitExtraFields* property. Set the *Name* column to **STDREENTRY** and set the *Value* column to the correct literal value. Review your WebRoutine NavMain RDMLX code to find the required values.
8. Save your changes.

## Step 4. Test and Understand the WebRoutine

In this step, you will test the WebRoutine, to understanding how it works.

1. NavMain should still be open in the *Design* view, if not, open it.
2. Click the *Execute* button on the toolbar to run the WebRoutine in the browser.
3. Test the buttons.

You will notice that all three searches function identically. The only way to differentiate them is the messages they generate.



The screenshot shows a web form titled "LANSA Advanced Software Made Simple" with a sub-header "Navigation Home". The form includes a dropdown menu with "A3564" selected, followed by "Search", "Transfer Search", "Call Search", and "Clear" buttons. Below these are input fields for "Employee Surname", "Employee Given Name(s)", "Street No and Name", "Suburb or Town", "State and Country", "Post / Zip Code" (with a "0" in the first box), and "Home Phone Number".

Also notice the message *This message will not be displayed*, is not displayed. At the time of a TRANSFER, control is passed to another WebRoutine and the current WebRoutine is terminated.

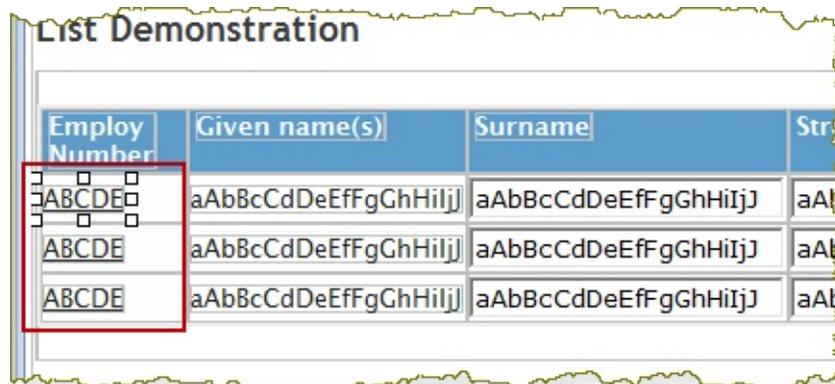
With a CALL, the current WebRoutine will still be executing and waiting for control to be returned from the WebRoutine it had called.

## Step 5. Add Weblet to a List

In this step, you will use the WAM created in exercise WAM015 (iiiWorkingLists) and add an Anchor weblet to the EMPNO field in the list. You will set up this link to call the search in iiiNavigation.

If you leave iiiNavigation open in the editor, when setting the *on\_click\_xxxx* properties in the following steps, you will be able to select the WAM name and WebRoutine name from a dropdown list.

1. Open WAM iiiWorkingLists in the LANSa editor.
2. Open ListMain in the Design view.
3. This step will add an anchor weblet to the list and configure its properties:
  - a. From the *Weblet Templates tab*, drag and drop the *Anchor* weblet onto the EMPNO field.



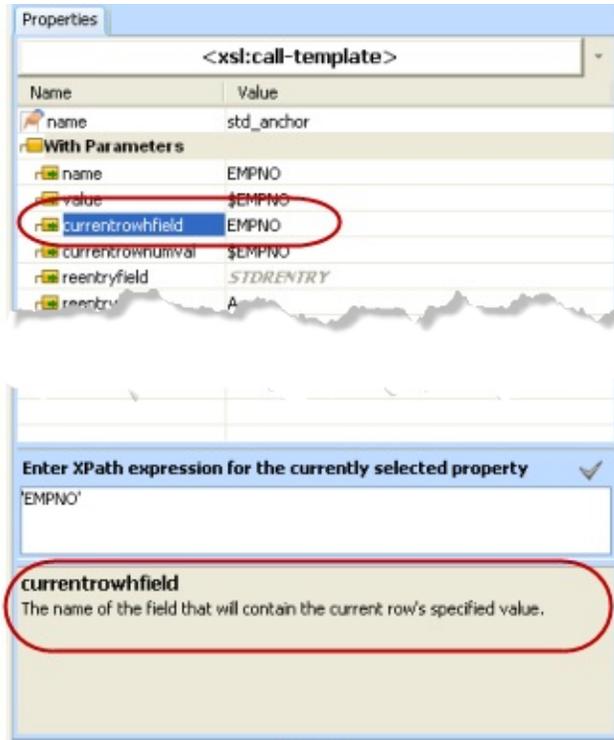
Employ Number	Given name(s)	Surname	Str
ABCDE	aAbBcCdDeEfFgGhHiIjJ	aAbBcCdDeEfFgGhHiIjJ	aA
ABCDE	aAbBcCdDeEfFgGhHiIjJ	aAbBcCdDeEfFgGhHiIjJ	aA
ABCDE	aAbBcCdDeEfFgGhHiIjJ	aAbBcCdDeEfFgGhHiIjJ	aA

Notice the anchor is applied to every row in the table.

- b. Set the *currentrowhfield* property to **EMPNO**.
- c. Set the *currentrownumval* property to **\$EMPNO**.
- d. Set the *reentryvalue* property to **A**.
- e. Set the *on\_click\_wamname* property to **iiiNavigation**.
- f. Set the *on\_click\_wrname* property to **NavMain**.

This will call NavMain with a reentry value of 'A' (remember, that is the local search) and will also pass the employee number of the clicked link.

**Note:** You can display help text for weblet properties:



**Currentrowfield** is the name of the field that will contain the current row's specified value.

**Currentrownumval** is the value of the field specified in the currentrowfield property. To specify a field value in a list use the syntax \$FIELDNAME. These properties enable the WebRoutine called via the on\_click\_wrname property to process values for the current row.

4. Save and run your modified WebRoutine LISTMAIN in a browser. Test the link to iiiavigation / NavMain. The employee details for the employee number selected in iiiWorkingLists should be displayed.

You can use the browser back button to return to iiiWorkingLists.

## Summary

### Important Observations

- You can navigate to WebRoutines in the RDMLX as well invoking them from the web page.
- TRANSFER leaves the current WebRoutine.
- CALL returns control to the calling WebRoutine after completion.
- A field's input box can be replaced with a weblet, to display the field values differently, just by dragging and dropping.
- The *Combo box* weblet can be used to display a dropdown list, which uses specified list entries to populate the dropdown items.
- The list that populates the *Combo box* is specified via weblet properties, via the *Details View*.
- The *Combo box* weblet can be made to invoke a WebRoutine on selection change.

### Tips & Techniques

- Not all lists have to be represented as browse lists. You can use a list for other purposes, as in the case of this page, which uses EMPLST to populate a dropdown list.
- Any field input boxes can be replaced with weblets, just by dragging and dropping a weblet onto them.

### What I Should Know

- How the TRANSFER and CALL commands work.
- There is more than one way to go about developing an application. Depending on the situation, certain methods work better than others. You should always consider your options and try to determine which method will work best for the given task.
- How to replace fields with weblets.
- How to attach lists to a weblet using its listname property.

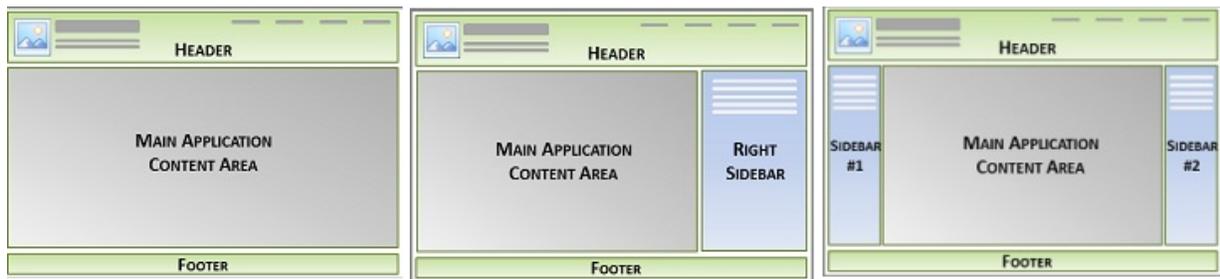
# WAM025 - Using the Layout Wizard

## Objectives

To create a WAM layout using the Layout Wizard.

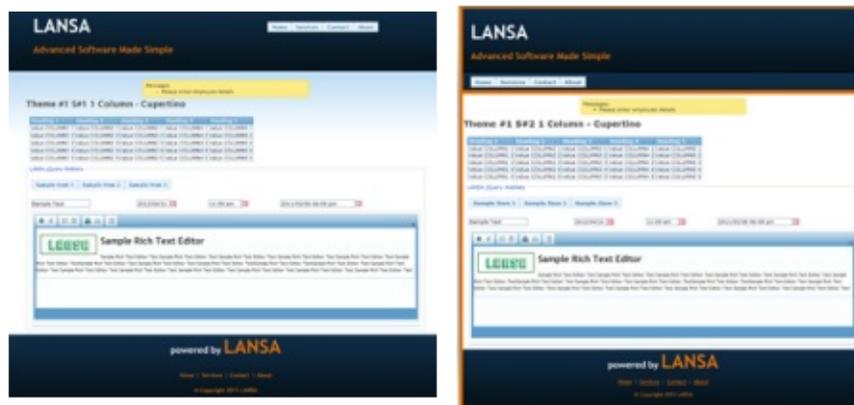
The Web Application Layout Manager Wizard enables you to define your own layout that can then be used for each WAM you create. The wizard can create a layout based one of three shipped designs which can adopt one of two styles. Each layout can also adopt a theme that controls the color scheme the layout will have. Once you have defined your own layout, it could be modified to meet your company's requirements.

Your layout may have one main area (the Main application Content Area) or include one or two sidebars as shown below:



A layout may adopt one of two styles

- Style1 has no border between areas
- Style2 defines a different color border between areas:



Themes allow you to select one of nine color schemes to apply to your layout. Three examples are shown below:



You can also choose between **fixed** or **fluid** layout width. The fluid layout is more flexible because its content area will include scroll bars if your content does not fit within the space available. For example some of your web pages may include lists with a large number of columns.

Once you have created a layout you could start to modify its appearance or content. For example, replace the fixed text such as 'LANSA Advanced Software Made Simple'. You will look at layouts in more detail in [WAM110 - Create Your Own Layout Weblet](#).

To create your layout you will complete the following:

[Step 1. Use the Web Application Layout Manager Wizard.](#)

[Step 2. Execute the generated Demo WAM](#)

[Step 3. Examine the new layout](#)

[Summary](#)

## **Before you Begin**

In order to complete this exercise, you should have completed the following:

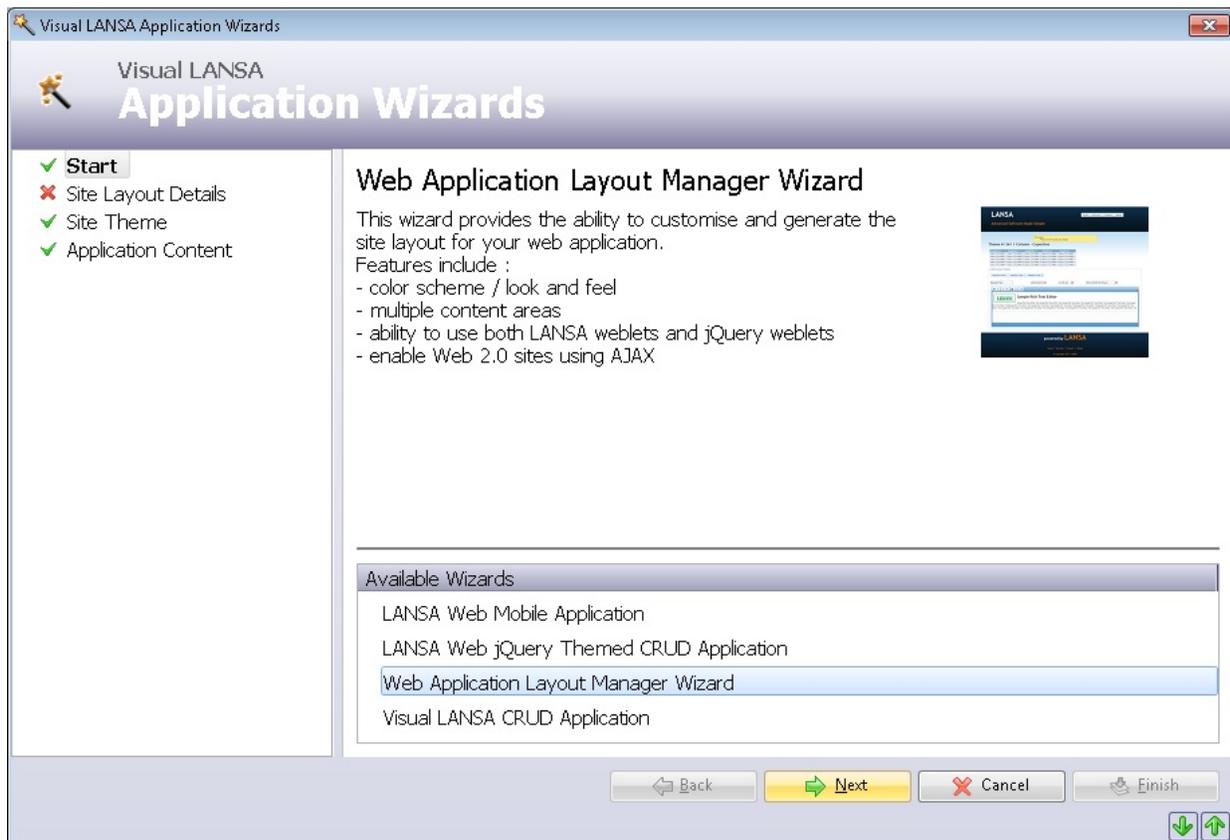
- [WAM005 - Create Your First WAM](#)
- [WAM010 - Using WEB\\_MAPs](#)
- [WAM015 - Working Lists](#)
- [WAM020 - WAM Navigation](#)

## Step 1. Use the Web Application Layout Manager Wizard.

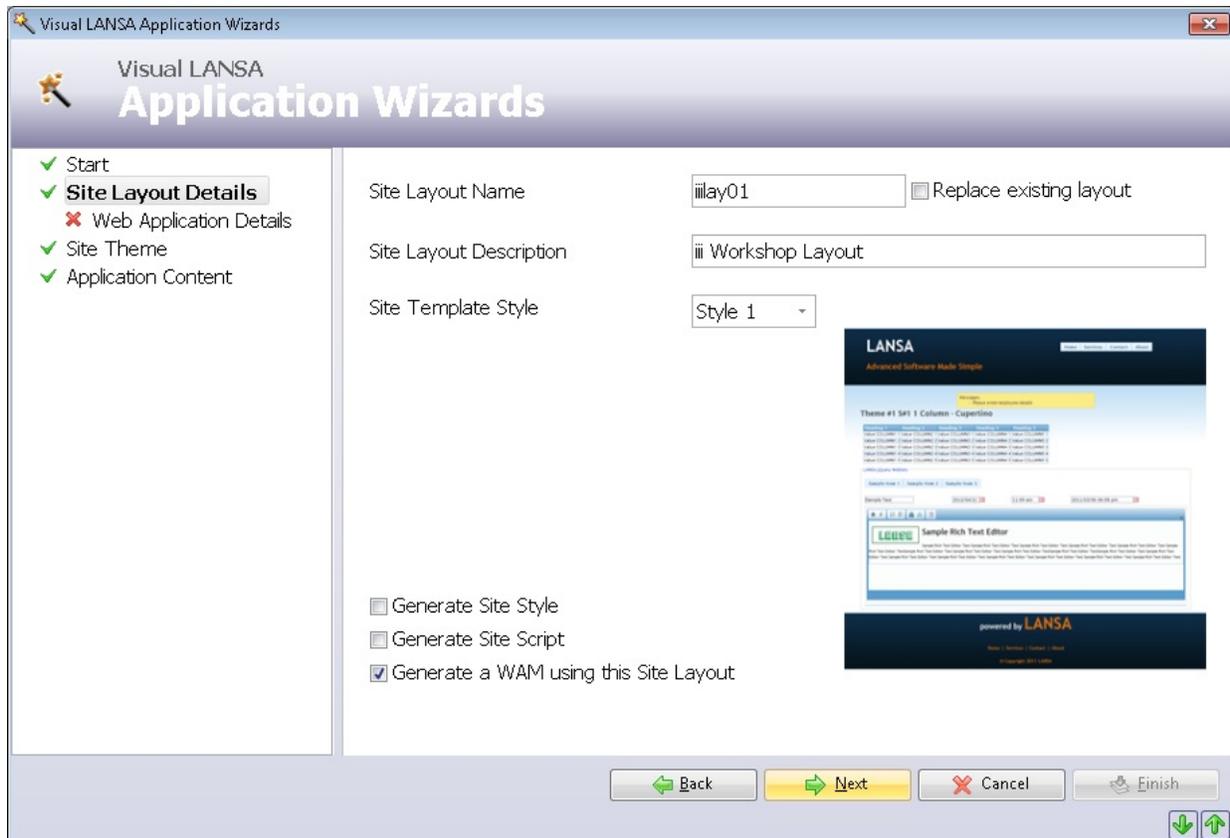
1. Start the wizard from the Tools ribbon in the *Utilities* grouping:



2. Select the *Web Application Layout Manager Wizard*.



3. Click *Next* to continue.
4. Enter the following *Site Layout Details*:
  - a. Site Layout Name: **iiilay01**
  - b. Site Layout Description: **iii Workshop Layout**
  - c. Generate a WAM using this *Site Layout*.

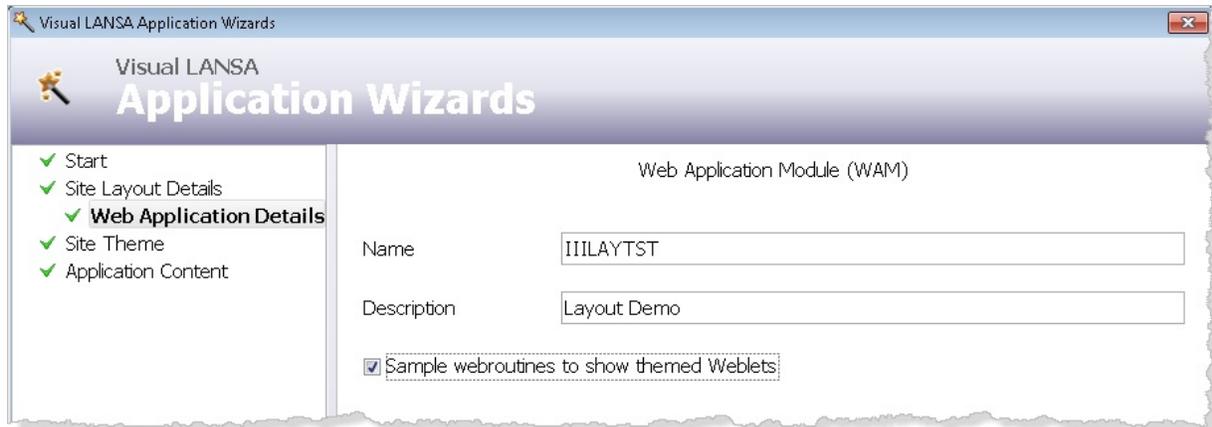


If *Generate Site Style* or *Generate Site Script* is selected, an *Application Images Directory* field will be displayed. The *Application Images Directory* will use an existing folder if entered. If a new folder name is entered this will be created in c:\Program Files (x86)\LANSA\Webserver\Images.

The new folder could be used to store application specific images.

Note that you would also need to later, set this folder up on the IBM i server.

- d. Click *Next*.
5. Enter the *Web Application Details*:
  - a. Name: **iiiLAYTST** for this exercise. In your own WAMs you may use a long name.
  - b. Description: **Layout Demo**
  - c. Select the *Sample WebRoutines to show themed Weblets* option.

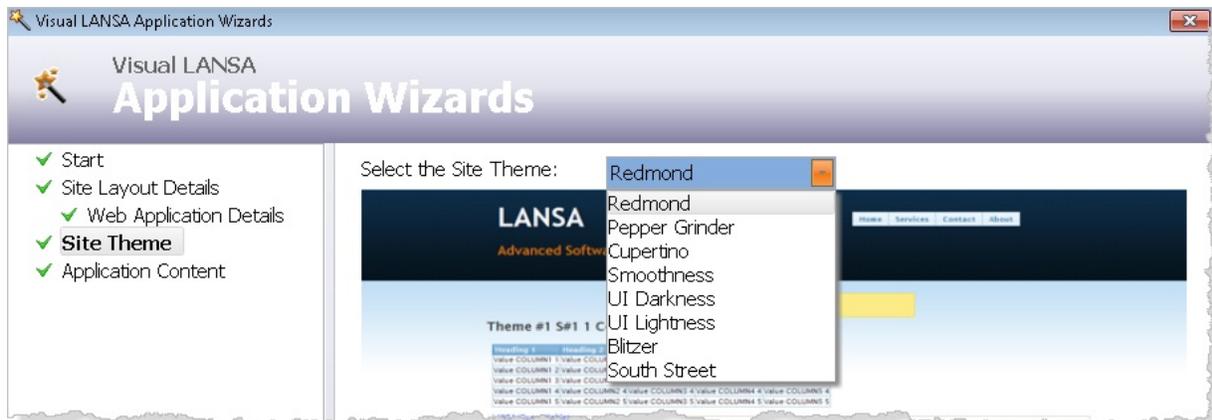


d. Click *Next*.

The Wizard creates a sample WAM for you.

6. Select one of the *Site Themes*.

The color scheme will be displayed when selected.

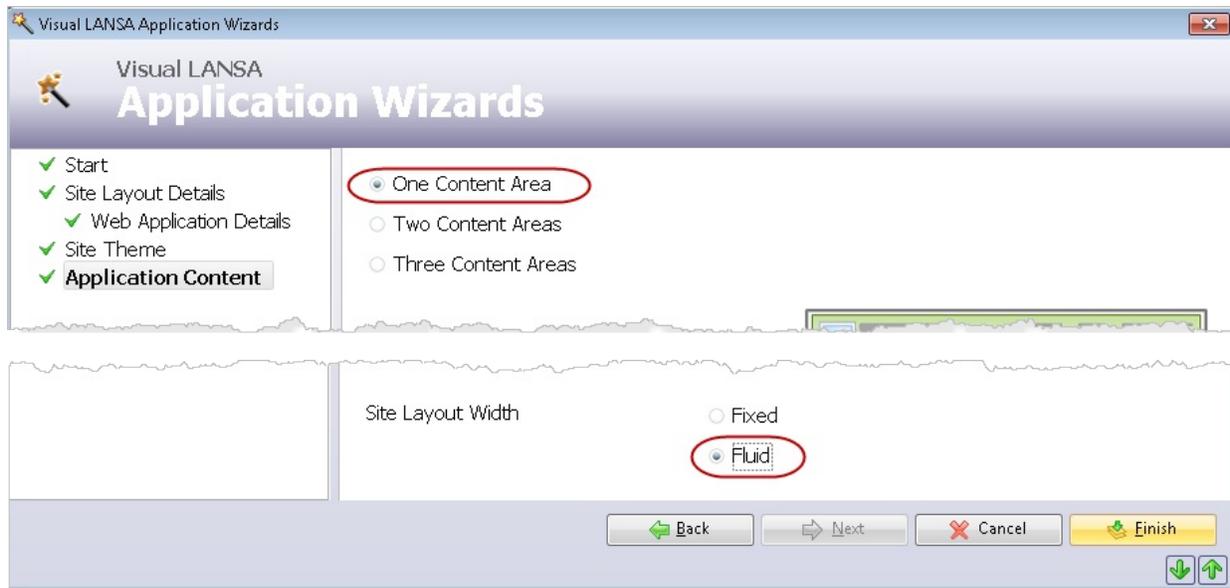


7. Click *Next*.

8. In the Application Content:

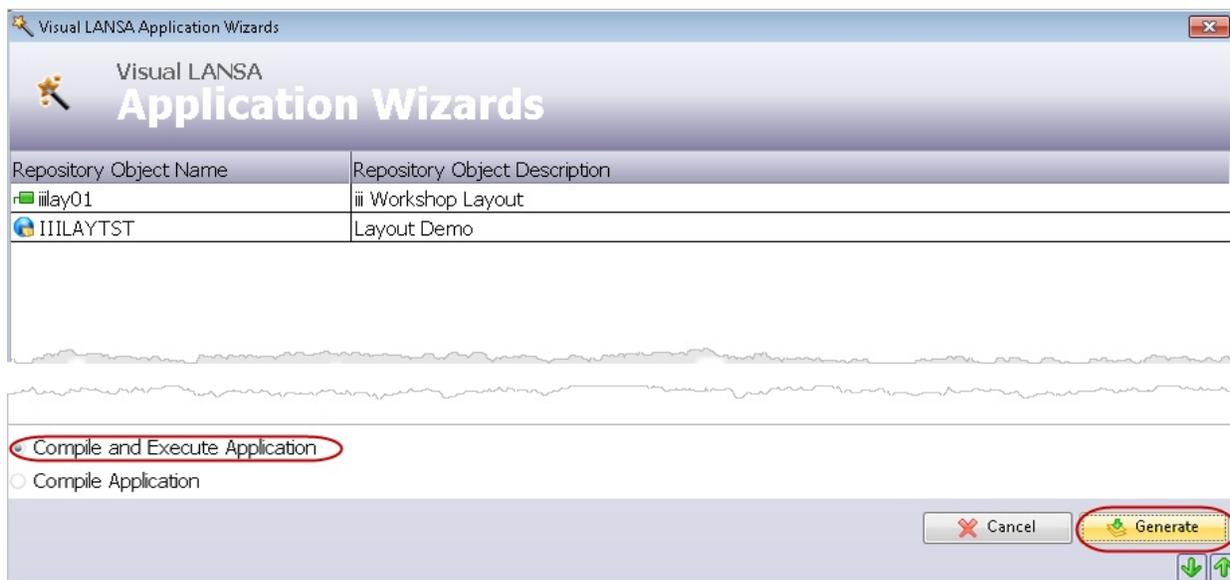
a. Select the *One Content Area*:

b. Select the **Fluid** *Site Layout Width* option.



c. Click *Finish*.

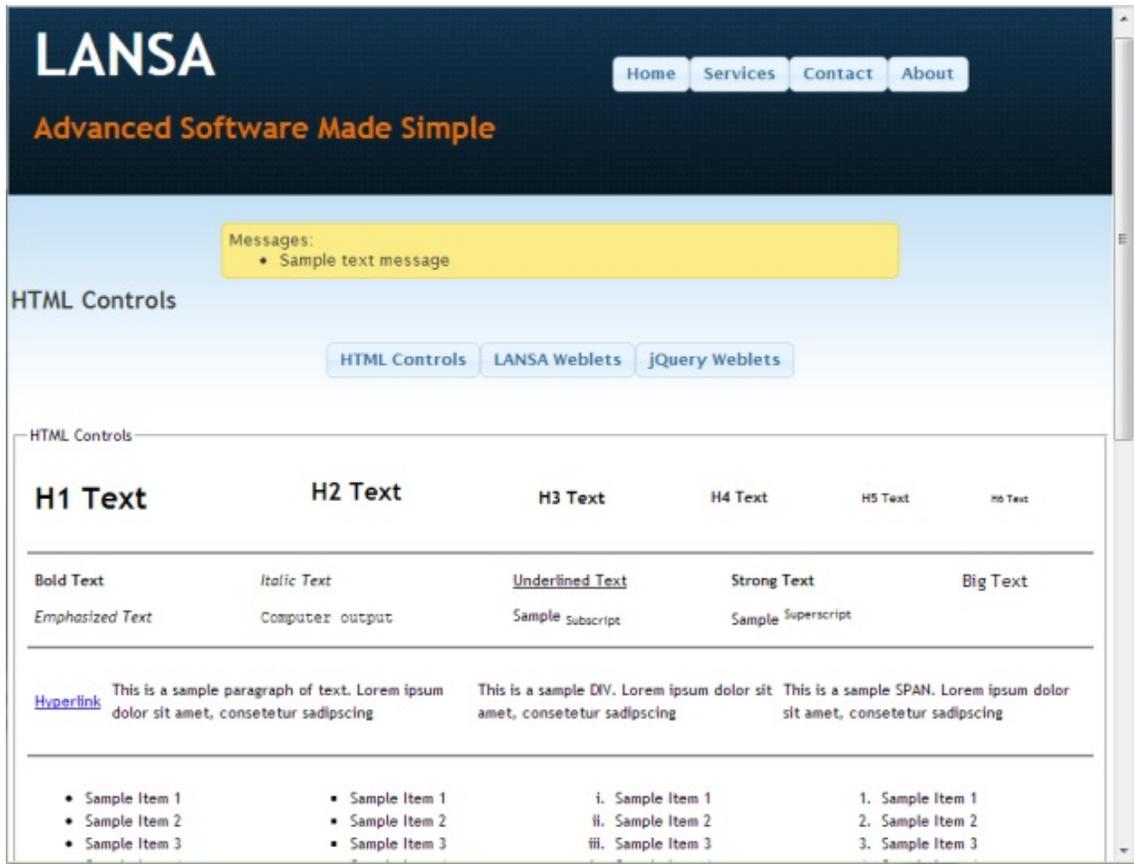
9. Select the *Compile and Execute Application* option.



10. Click *Generate*.

- First, your layout **iiilay01** is generated.
- Next, your WAM is generated together with its WAM layout which references **iiilay01**.
- Third, your WAM is opened in the browser.

**Note:** The appearance of your WAM will depend on the Theme that you selected in the Wizard questionnaire.



- The demo WAM contains three WebRoutines that demonstrate:
    - Basic HTML controls
    - LANSA weblets
    - jQuery enabled weblets
- The WebRoutines can be invoked from the menu at the top of the content area.

## Step 2. Execute the generated Demo WAM

1. Open your iii Demo Layout WAM (iiiLAYTST) in the editor so that you can review its RDMLX code as you are running the WAM. If it is not already started in the browser, run the SampleHTML WebRoutine from the Design view.

Note that this page contains sample standard HTML only. The WebRoutine SampleHTML contains no web\_maps or code except for a MESSAGE command.

2. Invoke the WebRoutine **SampleWeblets1** by selecting LANSA Weblets in the menu at the top of the content area.

This web page contains output for standard LANSA weblets. These have all been supported since WAMs were introduced into the Visual LANSA product in 2005. Note that they adopt the color scheme for your chosen Theme. For example the menu button, the list and tab pages.

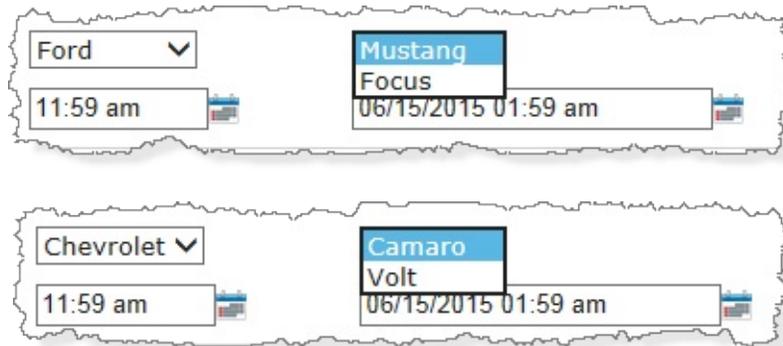
- a. Review the **SampleWeblets1** WebRoutine. This has a web\_map for the list that supports the tree view weblet (car manufacturers and models). The tree view list is populated by method routine BuildSampleTree.
  - b. Note that the content for all other controls on this page has been hard coded. For example, the weblet's *items* property defines the content for the combo box and list box.
3. Invoke the WebRoutine **SampleWeblets2** by selecting the jQuery Weblets option in the menu at the top of the content area.

This page demonstrates the new weblets that were introduced in V12 SP1 (June 2011). These weblets will all be used and explained in more detail in later exercises.

A number of these new weblets are AJAX enabled, which means a single control can be refreshed from the server. For example, the list that defines the content of a Dynamic select box can be refreshed by invoking a new type of WebRoutine.

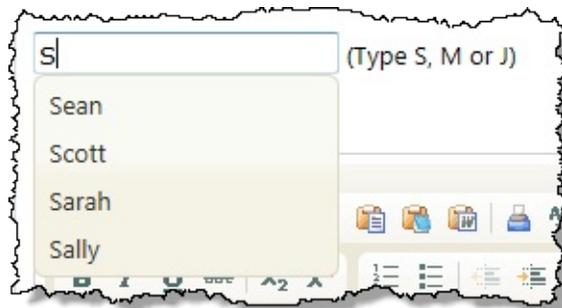
- a. Review the page content. Note that the weblets reflect the color scheme of your chosen Theme, including the more complex weblets such as the bar charts and pie charts. If possible review the appearance of other student's layouts who may have selected a different Theme.
- b. Notice that the dropdown lists for car manufacturer and car models are

dynamic.



When car manufacturer changes, the car models dropdown list is refreshed with a new list by calling a special WebRoutine. The rest of the page is unchanged.

- c. Type an appropriate letter into the AutoComplete input field.



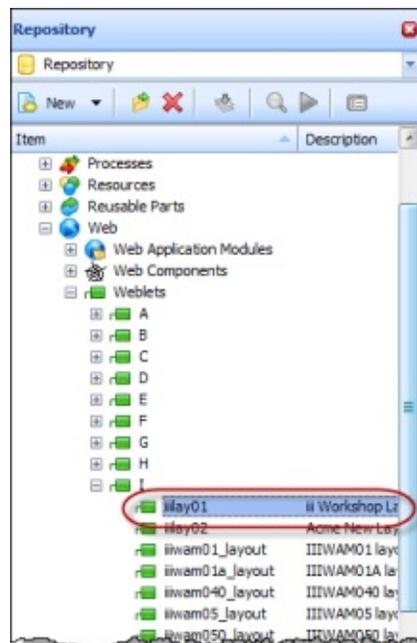
The list that is displayed, is also populated by a special response WebRoutine. Once again this is the only part of the web page that has been updated.

- d. Review the RDMLX code for the **SampleWeblets2** WebRoutine. Notice that it outputs lists for dropdown lists, charts and tree. These lists are built by invoking method routines.
- e. At this stage, we will leave an explanation of the special WebRoutines that handle the dynamic dropdown list and auto complete input field. These will be implemented in later exercises.

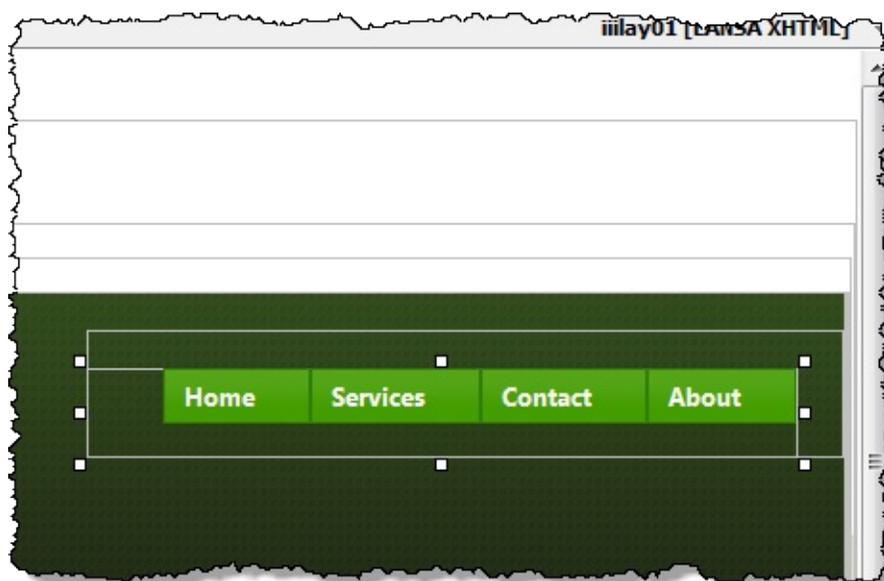
### Step 3. Examine the new layout

In this step you will find your new layout in the Repository and open it in the editor.

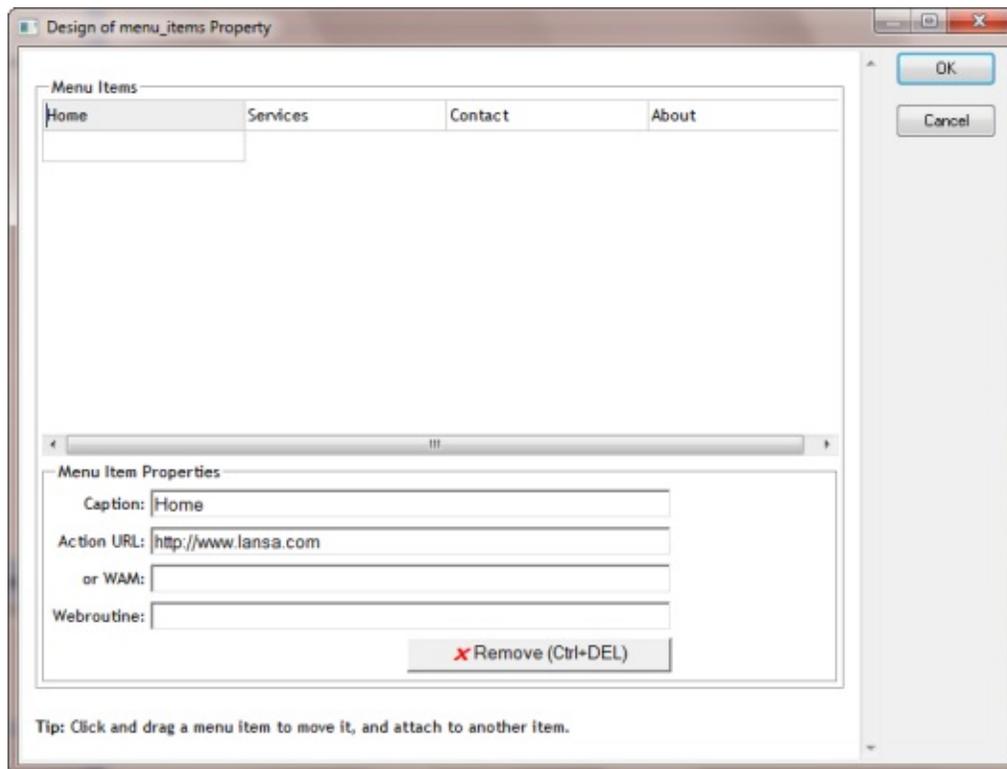
1. Layouts are special type of weblet. Expand the Web / Weblets group on the Repository tab to locate your layout. Double click on it to open it in the editor.



2. In the *Design* view, select the Menu Bar item.



3. On the *Details* tab, select the items value, and click on the Ellipsis  button to open the *Design of menu\_items Property* dialog.



4. Select each menu item and define its Action URL based on the following:

Home **<http://www.lansa.com>**

Services **<http://www.lansa.com/services/index.htm>**

Contact **<http://www.lansa.com/about/contactus.htm>**

About **<http://www.lansa.com/about/index.htm>**

5. Click OK to close the dialog and save your layout.
6. Open your iii Demo Layout WAM (iiiLAYTST) in the editor and run the SampleHTML WebRoutine in the browser from the *Design* tab. Test your menu items which should open the relevant pages within the LANSA web site. Use the browser back button to return to your application web page.

This illustrates, in one simple way, how your layout can be developed and all WAM layouts based on this standard layout will adopt changes made to the

common layout.

## **Summary**

### **Important Observations**

The layout wizard allows you to quickly and easily create a standard layout that can be adopted when creating your application WAMs

In a later exercise you will see how this layout can be modified to match your company requirements in more detail.

The fonts and color schemes used by your layout are controlled by Cascading Style Sheet files (CSS). In a later exercise you will see how these CSS files can be edited to meet your company requirements exactly.

### **Tips & Techniques**

Start your web application development by creating a standard layout and then specify this layout as each WAM is created.

This layout does not have to be complete in every respect. You can modify it later and all your WAMs will implement these changes.

### **What I Should Know**

How to use the Web Application Layout Manager Wizard, to create a layout.

## WAM030 - Employee Enquiry

### Objectives

- To create a simple web application
- The WAM prompts the user to enter an Employee number and then following a button press, displays Employee Details.



To achieve these objectives, you will complete the following:

Step 1. [Create Employee Enquiry WAM](#)

Step 2. [Create a Begin WebRoutine](#)

Step 3. [Open the Design View](#)

Step 4. [Add a Push Button Weblet](#)

Step 5. [Create WebRoutine: Details](#)

[Summary](#)

### Before You Begin

- In order to complete this exercise, you should have completed all the preceding exercises.

## **Step 1. Create Employee Enquiry WAM**

1. In the LANSAs Editor, click the New button and choose Web Application Module. The New WAM dialogue will appear.
2. In the New WAM dialogue box enter:
  - a. Name: iiiEmpEnquiry where iii are your initials
  - b. Description: Employee Enquiry
  - c. Layout Weblet: iiilay01 – the new layout you created in exercise WAM025
  - d. Select any suitable Framework, such as Personnel & Payroll
  - e. Click the Create button to create a new WAM

## Step 2. Create a Begin WebRoutine

In this step you will create the code for a new WebRoutine that will be used to enter the employee number. The WEB\_MAP statement will specify the fields that are passed in and out of the WebRoutine.

1. Immediately following the BEGIN\_COM, insert the following RDML code to create WebRoutine named Begin.

```
Webroutine Name(Begin) Desc('Select Employee')Endroutine
```

2. This WebRoutine will produce a web page with employee number as an input field. You will add a push button to invoke a second WebRoutine. The **Begin** WebRoutine will not require any incoming fields. It will have one outgoing field, EMPNO. Add the following WEB\_MAP statement to the WebRoutine.

```
Web_Map For(*output) Fields(#empno)
```

The EMPNO field will be used to enter the employee number to be retrieved and displayed.

Remember that by default, fields defined on a WEB\_MAP statement are input capable.

For further information about the WEB\_MAP statement, refer to the *LANSA Technical Reference Guide*.

**Note:** you can press F1 on any statement in the LANSAs editor to display help directly from the LANSAs Technical Reference Guide.

Your completed WebRoutine should appear as follows:

```
Webroutine Name(Begin) Desc('Select Employee')  
Web_Map For(*output) Fields(#empno)  
Endroutine
```

3. Compile your WAM.

### Step 3. Open the Design View

After compiling your WAM, you can edit the web pages for your WebRoutines.

1. In this case your WAM contains just one WebRoutine. If you select the Design tab, it will open the first WebRoutine in the Design view. You will use other methods to open a WebRoutine in the *Design* view later in this exercise.

Your web page should look like the following:



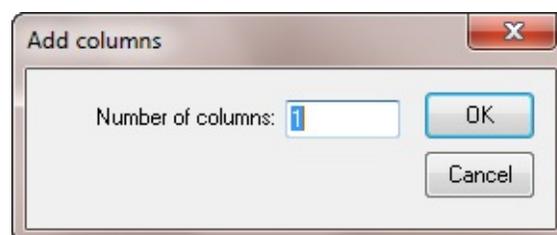
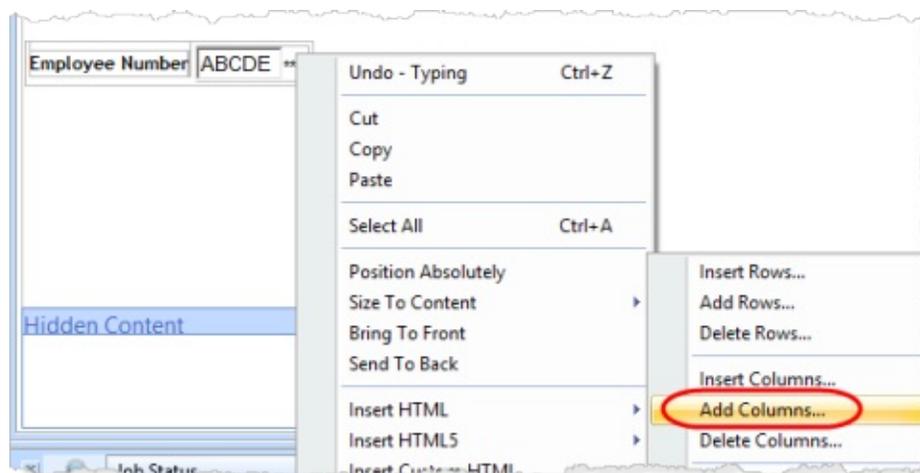
## Step 4. Add a Push Button Weblet

In this step you will add a column to the table containing employee number, and drag a push button weblet into the new cell.

1. Notice that the table is shown in the editor with a double line border. This is to make the table visible for editing. The table will not have a border when you run the WebRoutine in the browser, unless you have edited the table definition and defined a border.

If you click anywhere in this table, you can use the right mouse menu (also known as the context menu) to select the Table Items menu. Hint: Select the employee number input box and move the cursor right to position into the table cell.

Select the Add columns... option to add a column(s) to the right hand side of the table.

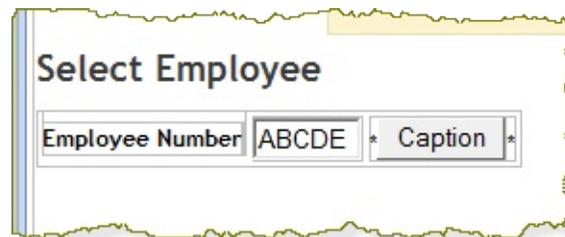


You should now see something like this:



Note that the new column contains characters \*\*. These are placeholders so that it is easy to drag and drop into this cell. You will later remove them.

2. On the *Favorites* tab, select the  **Weblet Templates** tab. Ensure that *Standard Weblets* is selected using the dropdown list at the top of this tab. Drag and drop a Push Button into the new cell at the right hand side of the table:



3. Use the keyboard cursor keys to position into the table cell that now contains the pushbutton and delete the two asterisks (they have done their job). For example select the push button and move right using the cursor key. You are now positioned in the table cell (you have the `<td>` tag selected) and can delete the \* character.
  - a. Set focus on the push button, select the  *Details* view to show the weblet's *Properties*.
  - b. Set the *caption* of the button to **Details**. **Note:** quotes are not required.
  - c. Set the *on\_click\_wrname* property to Details (note that you have not yet written a WebRoutine named 'Details').
  - d. Click the ellipsis button for the *submitExtraFields* property. Complete the *Design of submitExtraFields Property* dialog:

Select field Name **EMPNO**

Enter Value **EMPNO** and select *Field* radio button

Click *OK*.

Your Design view should now look something like this:



4. Save your changes. In the *Design* view you are actually editing an XSL document. It is good practice to save your work regularly.

## Step 5. Create WebRoutine: Details

1. Select the Source tab to return to your WAM's RDMLX code and add a new WebRoutine named **Details**.

This WebRoutine needs to:

receive a value for the EMPNO field from the browser

retrieve the employee record using a FETCH command from the file PSLMST,

display values for EMPNO, GIVENAME, SURNAME, ADDRESS1,

ADDRESS2, ADDRESS3, POSTCODE, PHONEHME, PHONEBUS,

DEPARTMENT, SECTION, SALARY, STARTDTE and TERMDATE. All these

fields should be mapped for output, at this stage you are writing an enquiry WebRoutine.

**Hint #1** : Remember it's a good programming technique to use a GROUP\_BY to define a set of fields. You can also use a GROUP\_BY to map fields into or out of the WebRoutine. Remember to set all fields in the group as \*OUTPUT to ensure they are displayed and cannot be changed (you can also use \*OUT to save on typing).

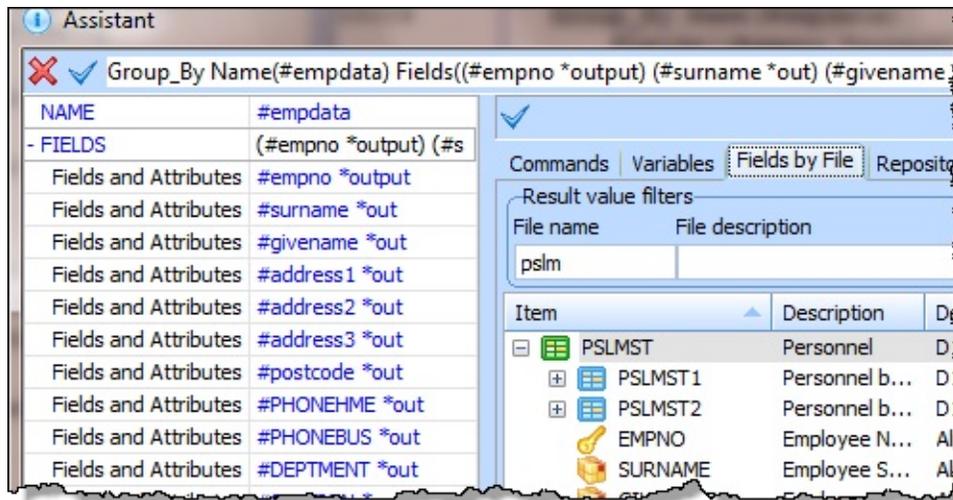
**Hint #2**: Always create a GROUP\_BY command with the *F4 Command Assistant*. This dialog allow you to quickly select fields from a file definition. You can then also define the \*out attribute against each field, while using this dialog.

To do this, click in one of the Fields and Attributes values, use the cursor key to move right, then type space, followed by \*out.

**Hint #3** : Think about what fields are "input" to the WebRoutine and which fields are "output" from the WebRoutine (to be displayed on the web form). This affects the For() parameter required on the web\_map.

**Hint #4**: As with all database programming, consider how to handle an error condition. For example you could use an IF\_STATUS to display an error message if the Employee record is not found. The LANS A I/O status is returned as field IO\$STS and the IF\_STATUS command compares with this field.

**Hint #5**: In a WAM, a validation error on an I/O command will branch to the EndRoutine, unless you have written VAL\_ERROR(\*next).



Your WAM code should now look like the following:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iilay01')
Group_By Name(#empdata) Fields((#empno *output) (#surname *out) (#givenname *out)
WebRoutine Name(begin)
Web_Map For(*output) Fields(#empno)
Endroutine
Webroutine Name(details) Desc('Employee Details')
Web_Map For(*output) Fields(#empdata)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno) Val_Error(*none)
If_Status Is_Not(*OKAY)
Message Msgtxt('Employee not found')
Endif
Endroutine
End_Com
```

2. Compile your WAM. The changes you made to the "Begin" WebRoutine web page will be retained. XSL and XML will be built for your new WebRoutine only, as part of the compile.
3. To open the Design view for the Details WebRoutine, use the  icon in the source code. Your web page should look like the following:

Employee Details	
Employee Number	ABCDE
Employee Surname	aAbBcCdDeEfgGhHijJ
Employee Given Name(s)	aAbBcCdDeEfgGhHijJ
Street No and Name	aAbBcCdDeEfgGhHijJkKlLm
Suburb or Town	aAbBcCdDeEfgGhHijJkKlLm
State and Country	aAbBcCdDeEfgGhHijJkKlLm
Post / Zip Code	123456
Home Phone Number	ABCDEFHGHIJKLMNO
Business Phone Number	ABCDEFHGHIJKLMNO
Department Code	ABCD
Section Code	AB
Employee Salary	123,456,789.12
Start Date (DDMMYY)	123456
Hidden Content	

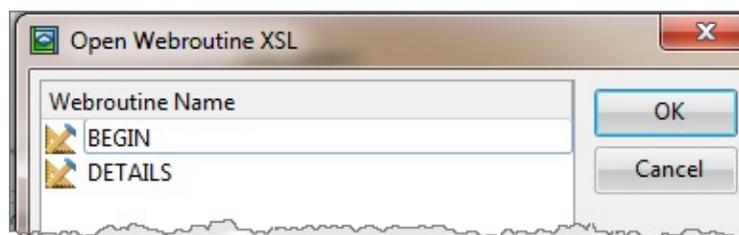
- The fields DEPARTMENT and SECTION may have a dropdown weblet visualization defined in the Repository. In this case their value will not be shown when they are output fields. You will be using field visualizations in a later exercise. If their value is not currently shown, select each in turn and use the context menu option *Change to Output Field*. This will display these fields as normal output fields.

Save your changes using the *Save*  button on the toolbar.

- On the *Design* ribbon click on the *Open WebRoutine* button.



Select the *Begin* WebRoutine and click *OK* to display it in the *Design* view.



- On the toolbar, click the  *Execute* icon to execute the WAM in the browser. Your web page should look like the following:



Enter an Employee number (for example, A0070, A0090 or A1234) and *click* the *Details* button.

The details web page should then be displayed.



To select another employee, return to the Select Employee by using the browser back button. In later exercises, you will learn how you to add a *Return* button on the Details page to return to the Begin WebRoutine.

7. You have completed this exercise.

## Summary

### Important Observations

- The WEB\_MAP statement allows you to specify which incoming and outgoing fields the WebRoutine maps between the web page and the WebRoutine. The WEB\_MAP statement's FOR() selector specifies whether the fields are mapped as an incoming (\*INPUT), outgoing (\*OUTPUT), or both (\*BOTH).
- Only fields in a WEB\_MAP statement with a FOR(\*OUTPUT) or FOR(\*BOTH) can be placed on the web page.
- Acceptable field attributes are \*INPUT and \*OUTPUT (also \*in and \*out). If unspecified, the default is \*INPUT. These attributes determine whether the field accepts input, as input text box, or only displays output.

#### Note an important distinction here:

The WEB\_MAP FOR() parameter determines whether fields and lists are mapped into the WebRoutine, out of the WebRoutine or in both directions.

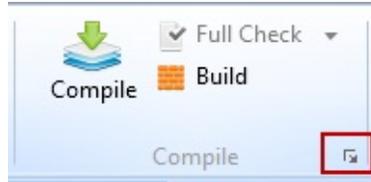
A field's \*INPUT or \*OUTPUT attribute determines whether the field will be input capable on the page (the default) or output only.

#### Note also that fields with an \*output attribute cannot be mapped back into a WebRoutine.

- You can open a WebRoutine in the *Design* view using the  icon in the RDMLX source
- You can drag and drop fields and lists marked FOR(\*OUTPUT) or FOR(\*BOTH) onto your page at any time from the  Webroutine Output tab.

### Tips & Techniques

- Weblet properties can be assigned string literals or XPath expressions. The XSL processor then evaluates the XPath expression. XPath expressions accept general logical comparison operators <, >, !=, <=, >= etc., as well as mathematical operators \*, /, +, - etc.  
For a quick reference to XPath see:  
[www.mulberrytech.com/quickref/XSLT\\_1quickref-v2.pdf](http://www.mulberrytech.com/quickref/XSLT_1quickref-v2.pdf).  
For a more detailed reference, see [www.w3schools.com](http://www.w3schools.com).
- With the WAM open in the editor, you can compile from the *Home* ribbon or open the *Compile options* dialog from the *Menu* button on the *Compile* group on the *Home* ribbon.



- You can also use a context menu by selecting a WAM in the *Repository* tab or *Favorites/Last Opened* tab.
- For a new WebRoutine, it is not necessary to do a compile to generate the XSL, since a  Build also generates the XML / XSL, but without doing a full compile.
- **Note:** XSL can be generated for a *selected WebRoutine* using the context menu option, *WebRoutine: nnnnnnn / Generate XSL* in the LANSAs Editor.

## What I Should Know

- How to create a simple enquiry WAM.
- How to open the *Design* view for a specific WebRoutine.
- What is generated by a WAM compile.
- A WAM may contain more than one WebRoutine
- A WebRoutine name may be up to 20 characters long.
- There is one WAM layout generated for each WAM with the name xxx\_layout, where xxx = the WAM name.
- By default, fields are visualized as a label and edit box
- Fields are displayed in a table
- A WebRoutine may be called by a weblet such as a push button, via its on\_click\_wrname property.
- Field names or a group\_by may be used to define fields in a web\_map.

## WAM035 - An Employee Update WAM

### Objectives

To create an employee update WAM **iiiEmpUpdate – Employee Update**, starting from **iiiEmpEnquiry – Employee Enquiry**.

All the screen fields except employee number need to be made input capable, and an *Update* button will be added to the *Details* page.

The image shows two overlapping screenshots of a web application interface. The top-left screenshot shows a summary view with the LANSA logo and the text 'Advanced Software Made Simple'. Below this, there is a field for 'Employee Number' with the value 'A0090' and a 'Details' button. The bottom-right screenshot shows the 'Employee Details' form, also with the LANSA logo and 'Advanced Software Made Simple' text. The form contains the following fields and values:

Field	Value
Employee Number	A0090
Employee Surname	BLOGGS
Employee Given Name(s)	FRED JOHN ALAN
Street No and Name	70 MAIN STREET
Suburb or dept	NEWTOWN NSW
State and Country	AUSTRALIA
Post / Zip Code	2220
Home Phone Number	344-2234454545
Business Phone Number	654 6475 X432
Department Code	FLT
Section Code	03
Employee Salary	20,045.91
Start Date (DDMMYY)	3/08/92
Termination Date (DDMMYY)	000000

An 'Update' button is located at the bottom right of the form.

To achieve these objectives you will complete the following:

Step 1. Create WAM **iiiEmpUpdate – Employee Update**

Step 2. Compile your WAM and complete the Details web page

Step 3. Test the Employee Update WAM

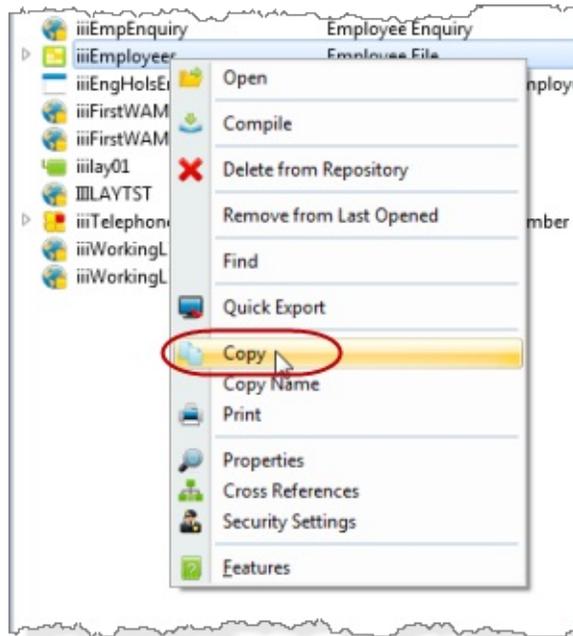
Summary

**Before You Begin**

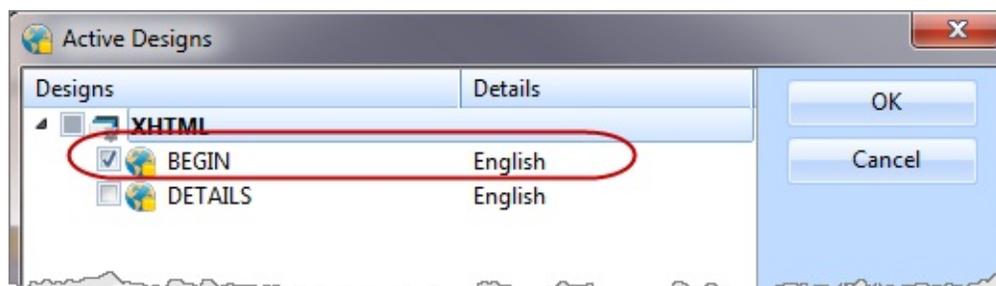
- In order to complete this exercise, you should have completed all the preceding exercises.

## Step 1. Create WAM iiiEmpUpdate – Employee Update

1. Select WAM iiiEmpEnquiry – Employee Enquiry on your Favorites / Last Opened tab and use the context menu to *Copy* it to create iiiEmpUpdate – Employee Update.



On the *Active Designs* dialog, select only the **Begin** WebRoutine:



In the Active Designs you selected only the BEGIN WebRoutine. This will copy the XML and XSL for this WebRoutine only. In the new WAM iiiEmpUpdate you are redesigning the DETAILS web page, making most of the fields input capable. When you compile iiiEmpUpdate having made your RDMLX changes to the DETAILS WebRoutine, the correct start position will be generated for the new DETAILS web page.

2. Change the GROUP\_BY so that all fields are input capable except for EMPNO, currently they are all displayed for output only. Once again use the

*F4 Command Assistant* to make these changes.

3. You need to ensure that the fields are both sent to the web page and received back from it, since this time you are displaying and updating employee fields. Remember that fields in a WEB\_MAP are by default mapped with an \*INPUT attribute. In order to display and update employee data, your fields will need to be mapped For(\*both).

4. The Details WebRoutine now needs to performs two roles. It FETCHs a record when invoked from the Begin WebRoutine, and UPDATEs the record when re-entered via an Update button. The field STDREENTRY will be used to control which logic to perform.

**Note #1:** STDREENTRY is a single character field which is defined as the default field returned by a number of weblets. This is simply a convention and you may prefer to replace it with a longer field which supports a more meaningful input values such as, ENQUIRE, UPDATE or DELETE.

**Note #2:** You should now map field STDREENTRY For(\*both) as a hidden field.

5. Modify the **Details** WebRoutine using a CASE / ENDCASE loop for field STDREENTRY, so that when the **Details** WebRoutine is called from the **Begin** WebRoutine (returning a STDREENTRY value of M) the employee record is fetched. When the **Details** WebRoutine is re-entered from the **Details** web page a value of U should perform an UPDATE to the employee file. The *Update* push button will return STDREENTRY with a value of 'U'.

The Employee Number (EMPNO) should be output on the Details page so that the key field cannot be changed. An output field on the web page cannot be mapped back into the next WebRoutine. Of course the WAM needs the Employee Number to perform the update.

This is a common situation with all web applications. One solution is to output a hidden work field containing the Employee Number and use this to perform the update.

6. Define a work field EMPNOW based on EMPNO. Map field EMPNOW for \*both as a \*hidden field. You could map both STDREENTRY and EMPNOW for all WebRoutines by defining this map at the WAM level (that is immediately following the **Define\_Com**). WEB\_MAPs defined at the WAM level are global and apply to all WebRoutines.

7. In the *Details* webroutine, when called from the *Begin* webroutine, set EMPNOW to the value of EMPNO.

8. When the *Details* WebRoutine is called by the Update button, set EMPNO to the value of EMPNOW.
9. Consider how to add logic to the Details WebRoutine to return to the Begin page if the update is successful.  
Hint: remember the TRANSFER command that you implemented in an earlier exercise.
10. Your WAM should now look something like the example following. The changed and new code compared with iiiEmpEnquiry is shown in red.

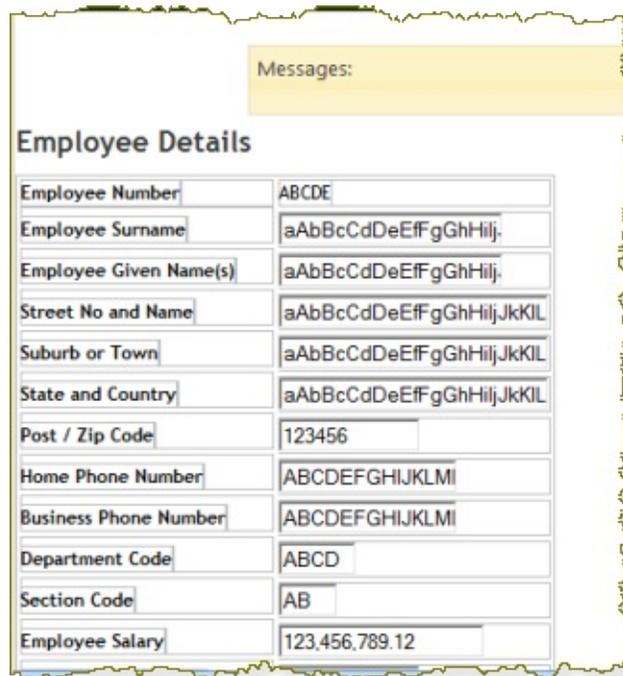
```

Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
Group_By Name(#EMPDATA) Fields((#empno *out) #SURNAME #GIVE
Define Field(#EMPNOW) Reffld(#EMPNO)
Web_Map For(*BOTH) Fields((#STDREENTRY *HIDDEN) (#EMPNOW *H
WebRoutine Name(begin)
Web_Map For(*output) Fields(#empno)
Endroutine
WebRoutine Name(Details) Desc('Employee Details')
Web_Map For(*BOTH) Fields(#EMPDATA)
Case (#STDREENTRY)
When ('= M')
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO) Val_Ei
If_Status Is_Not(*OKAY)
Message Msgtxt('Employee not found')
Transfer Toroutine(Begin)
Endif
Change Field(#EMPNOW) To(#EMPNO)
When ('= U')
#EMPNO := #EMPNOW
Update Fields(#EMPDATA) In_File(PSLMST) With_Key(#EMPNO) Val_
If_Status Is(*OKAY)
Transfer Toroutine(Begin)
Endif
Endcase
Endroutine
End_Com

```

## Step 2. Compile your WAM and complete the Details web page

1. Compile your WAM and open the Begin WebRoutine in the Design view. Notice that it is a copy of the web page design from WAM iiiWAM030, that you included in the Copy process. The Details push button is already defined.
  - a. Select Details push button and on the *Details* tab, click on the Ellipsis button for the *submitExtraFields* property to display the *Design of...* dialog. Select **STDREENTRY** in the *Name* column and enter **M** as a literal in the *Value* column
  - b. Click *OK* to close the dialog and confirm the change.
2. Open the **Details** WebRoutine in the Design view. Your web page should look like the following:



Employee Details	
Employee Number	ABCDE
Employee Surname	aAbBcCdDeEfGhHij.
Employee Given Name(s)	aAbBcCdDeEfGhHij.
Street No and Name	aAbBcCdDeEfGhHijJkKIL
Suburb or Town	aAbBcCdDeEfGhHijJkKIL
State and Country	aAbBcCdDeEfGhHijJkKIL
Post / Zip Code	123456
Home Phone Number	ABCDEFGHijklmI
Business Phone Number	ABCDEFGHijklmI
Department Code	ABCD
Section Code	AB
Employee Salary	123,456,789.12

3. As before the Department and Section fields may be defined in the Repository with a dropdown weblet visualization. You will be using these in a later exercise. For now you will make them normal input fields. To do this, select the Department dropdown list and using the context menu, select *Replace with Input Field*. Repeat this step for the Section field.
4. Select within the table containing the fields, and use the context menu Table Items to Add Row.... A row will be added to the bottom of the table.
5. Select the bottom right hand cell in the table and use the Detail tab to set its

*align* property to right. Drag and drop a push button weblet into this bottom right hand cell. Ensure that the button is selected, and use the Detail tab set up the button properties as follows:

<b>Property</b>	<b>Value</b>
caption	<b>Update</b>
on_click_wname	<b>Details</b>
submitExtraFields	<b>Field Name: STDREENTRY</b> <b>Literal Value: U</b>

**Note:**

- a. Select the *on\_click\_wname* property and set its value from the dropdown list.
  - b. Then use the *Ellipsis* button for the *submitExtraFields* value, to open the *Design of...* dialog.
6. Save your changes.

Your page should now look like the following:

## Employee Details

Employee Number	Value EMPNO
Employee Surname	ABCDEFGHIJKLMNOPS
Employee Given Name(s)	ABCDEFGHIJKLMNOPS
Street No and Name	aAbBcCdDeEfGhHiIjJkKlLm
Suburb or dept	aAbBcCdDeEfGhHiIjJkKlLm
State and Country	aAbBcCdDeEfGhHiIjJkKlLm
Post / Zip Code	123456
Home Phone Number	ABCDEFGHIJKLMNO
Business Phone Number	ABCDEFGHIJKLMNO
Department Code	ABCD
Section Code	VALU
Employee Salary	123,456,789.12
Start Date (DDMMYY)	12/34/56
Termination Date (DDMMYY)	123456
	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
	<input type="checkbox"/> Update <input type="checkbox"/>
	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

### **Step 3. Test the Employee Update WAM**

1. On the *Design* ribbon use the *Open Webroutine* button to open the *Begin WebRoutine* in the *Design* view.
2. Use the  *Execute* button on the *Home* ribbon to run it in the browser. Ensure that you can display and update an employee. When your update is successful you should be returned to the *Begin* web page.
3. Try to perform an invalid update. For example, make the surname field blank. Note that the *Details* web page is redisplayed and validation errors are displayed in the message area at the top of the page.

## Summary

### Important Observations

- Fields defined in a WEB\_MAP will be input capable unless the field has an \*output attribute
- The field STDREENTRY is the default field returned by many weblets such as a hyperlink, a check box or a radio button.
- Push buttons are may return a number of fields via the *submitExtraFields* property. You must use the Ellipsis button for this property and set up the field(s) and values(s) to be returned in the *Design of...* dialog. Always specify the *on\_click\_wrname* property before using the *Design of...* dialog.
- WebRoutines often use a CASE loop for field STDREENTRY to handle different execution logic.

### Tips & Techniques

- Web applications often need to use a work field to pass back a value for \*output key fields, such as employee number in this example
- Fields may be changed from input to output in the Design view
- You can open a WebRoutine in the Design view from the RDML source editor using the  Open WebRoutine icon. See example following:

```
Webroutine Name(begin)   
  Web_Map For(*output) Fields(#empno)  
Endroutine
```

### What You Should Now Know

- You can set *on\_click\_wrname* by selecting from a dropdown list – provided the WebRoutine is defined in your RDML code.
- How to code a simple WAM to perform display/update logic.
- How to use the TRANSFER command to invoke a WebRoutine.

## WAM040 - Add dropdown lists for Department and Section

### Objectives

The fields DEPARTMENT and SECTION are defined in two table files DEPTAB and SECTAB. Sections belong to a department and the SECTAB file is therefore keyed on DEPARTMENT and SECTION.

In this exercise you will replace the fields DEPARTMENT and SECTION with a combo box weblet, and link each weblet to a working list of values based on the table files DEPTAB or SECTAB. Because sections belong to a department, if the department selected is changed, the list of sections will need to be refreshed.

Your finished Details page will look like the following:



The screenshot shows the LANSA 'Employee Details' form. The header includes the LANSA logo and the tagline 'Advanced Software Made Simple'. The form contains the following fields:

Employee Number	A0090
Employee Surname	BLOGGS
Employee Given Name(s)	JOE
Street No and Name	12 River Lane
Suburb or Town	St Albans
State and Country	Herts
Post / Zip Code	2000
Home Phone Number	01727 554661
Business Phone Number	01727 8875643
Department Code	ADMINISTRATOR DEPT
Section Code	INTERNAL ADMIN SRV
Employee Salary	20,045.91
Start Date (DDMMYY)	3/08/92
Termination Date (DDMMYY)	0/00/00

An 'Update' button is located at the bottom right of the form.

To achieve these objectives, you will complete the following:

Review The [The Dynamic Select Box Weblet](#) and [Automatic Updating](#).

Step 1. Create [iiiWAM040 - iii Employee Update - Enhanced](#)

Step 2. [Add Dynamic Select Boxes to the Details Web Page](#)

Step 3. [Make the Sections Dropdown list dynamic](#)

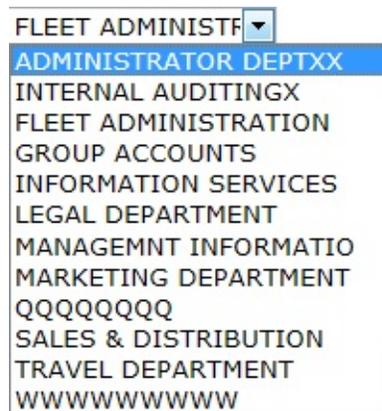
## Summary

### Before You Begin

In order to complete this exercise, you should have completed all the preceding exercises.

### The Dynamic Select Box Weblet

The dynamic select box weblet produces a dropdown list containing a list of values.



The list of values is usually provided by a working list output by the WebRoutine. Alternatively the weblet's items property may be used to define a hard coded list of values.

The list may contain two or three columns. The codefield defines a value to be returned when an entry is selected and the captionfield defines a description that will be displayed in the dropdown list. An optional third column defines a selector field that defines a group of values that should be displayed when a related field value changes.

The web\_map must define the working list for output and the list must be defined as using JSON format. For example:

```
Web_map For(*output) Fields((#deptdd *json))
```

JSON stands for JavaScript Object Notation. JSON is a lightweight data-interchange format. See <http://www.json.org/> for more information.

### Automatic Updating

The dynamic select box weblet is AJAX enabled. This means it is capable of invoking a specially defined "response" WebRoutine that rebuilds and outputs the list that supports the dropdown list.

The dynamic select box can monitor another field and automatically refresh itself whenever that field is updated. If the weblet has been filled using a

working list then you will need to create a JSON WebRoutine that will output a fresh copy of the working list. The weblet will call this WebRoutine each time it needs to refresh.

Only this JSON data is refreshed and the rest of the page is unchanged. This design can provide very responsive web applications. For example, this WebRoutine rebuilds a list of sections when department changes:

```
WebRoutine Name(UsectDD) Response(*JSON)
Web_Map For(*input) Fields(#deptment)
Web_Map For(*output) Fields((#sectdd *JSON))
#com_owner.buildDD2 I_Dept(#deptment) I_Sect(#section)
Endroutine
*
Mthroutine Name(BuildDD2)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Define_Map For(*input) Class(#section) Name(#i_sect)
#deptment := #i_dept
Clr_List Named(#sectdd)
Select Fields(#sectdd) From_File(sectab) With_Key(#deptment)
Add_Entry To_List(#sectdd)
Endselect
If (#i_sect *NE *blanks)
#section := #i_sect
Endif
Endroutine
```

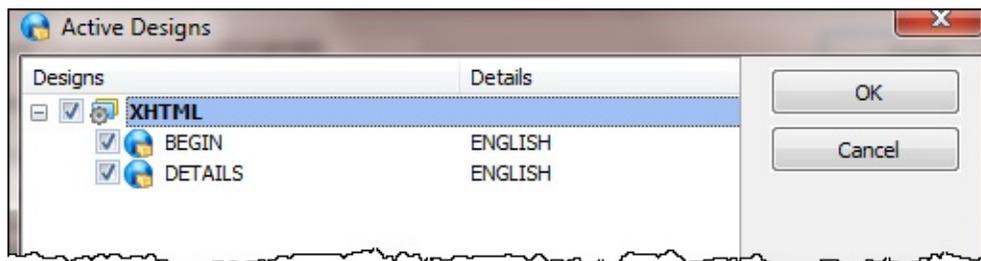
These properties set up the dynamic select box weblet to automatically call a WebRoutine to refresh:

- updateOnFieldChange* – the name of the field to be monitored
- updateWamName* – the name of the WAM to invoke. Only required if not the current WAM
- updateWrName* – the name of the WebRoutine to invoke
- updateFieldsToSubmit* – the name of one or more fields to be submitted when the monitored field changes.

## Step 1. Create iiiWAM040 - iii Employee Update - Enhanced

In this step you will create iiiEmpUpdate\_MK2 –Employee Update Enhanced, by copying WAM iiiEmpUpdate and then add logic to handle working lists for departments and sections.

1. Select WAM iiiEmpUpdate on the Favorites/Last Opened tab and use the context menu to copy it.
2. In the Create WAM dialog define your new WAM as – iiiEmpUpdate\_MK2 – Employee Update Enhanced.
3. In the Active Designs dialog, leave the Begin and Details WebRoutine checked. You are creating a copy of iiiEmpUpdate including its layouts (XSL).



4. In the new WAM, iiiEmpUpdate\_MK2, define working lists for department and section fields. Define the lists immediately following the Begin\_com. Your code should look like the following:

```
Def_List Name(#deptdd) Fields(#department #deptdesc) Type(*Working)
Entrys(*max)
Def_List Name(#sectdd) Fields(#section #secdesc) Type(*Working)
Entrys(*max)
```

5. Create a method routine BuildDD1 to clear and build the department list, DeptDD. This routine will require the following logic:
  - a. Map for input a variable based on DEPARTMENT. This will enable the routine to position the list to the current value of DEPARTMENT.
  - b. Clear the list DEPTDD
  - c. Add entries to DEPTDD for all records in file DEPTAB – Department code table
  - d. Reset DEPARTMENT to the input variable value.

Your completed code should look like the following:

```
Mthroutine Name(BuildDD1)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Clr_List Named(#deptdd)
Select Fields(#deptdd) From_File(deptab)
Add_Entry To_List(#deptdd)
Endselect
#deptment := #i_dept
Endroutine
```

6. Create a method routine BuildDD2 to clear and build the sections list, SectDD. This routine will require the following logic:
  - a. Map for input a variable based on DEPARTMENT. This will enable the routine to rebuild the list of sections for the current value of DEPARTMENT.
  - b. Map for input a variable based on SECTION. This will enable the routine to position the list to the current value of SECTION.
  - c. Set DEPARTMENT to the value of the input variable.
  - d. Clear the list SectDD
  - e. Add entries to the list from the SECTAB – Section code table, using DEPARTMENT as key.
  - f. If the passed variable for SECTION is not blank, set SECTION to the value of the input variable

Your completed code should look like the following:

```
Mthroutine Name(BuildDD2)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Define_Map For(*input) Class(#section) Name(#i_sect)
#deptment := #i_dept
Clr_List Named(#sectdd)
Select Fields(#sectdd) From_File(sectab) With_Key(#deptment)
Add_Entry To_List(#sectdd)
Endselect
If (#i_sect *NE *blanks)
#section := #i_sect
Endif
Endroutine
```

7. In this step you will make changes to your existing **Details** WebRoutine.
  - a. Add the lists DeptDD and SectDD to the web\_map with a JSON attribute. Your code should look like the following:

```
Web_Map For(*BOTH) Fields(#EMPDATA (#deptdd *json) (#sectdd *json))
```

- b. At the end of the Details WebRoutine invoke the new method routines, passing the value of DEPARTMENT and SECTION as required. Your WebRoutine should look like the following. Unchanged code isn't shown.

```
WebRoutine Name(Details) Desc('Employee Details')
Web_Map For(*BOTH) Fields(#EMPDATA (#deptdd *json) (#sectdd *json))
....
....
Endcase
#com_owner.BuildDD1 I_Dept(#department)
#com_owner.BuildDD2 I_Dept(#department) I_Sect(#section)
Endroutine
```

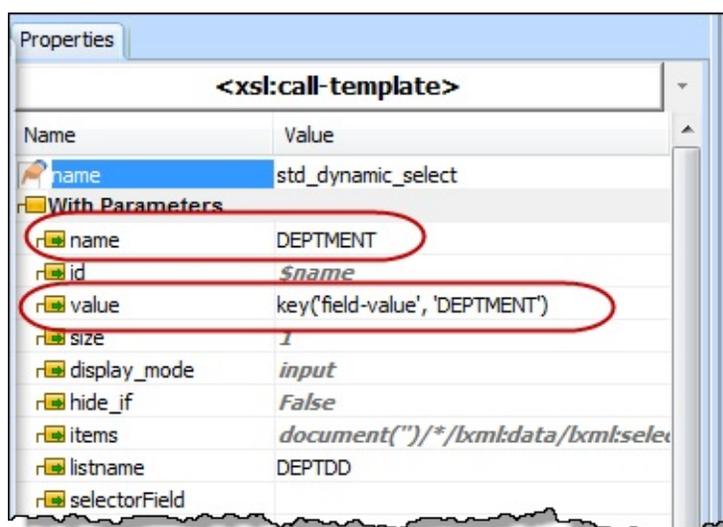
8. Compile your WAM.

## Step 2. Add Dynamic Select Boxes to the Details Web Page

In this step you will add dynamic select boxes to replace the department and section code fields, and test your WAM.

1. Open the **Details** WebRoutine in the *Design* view. Drag and drop a Dynamic Select Box weblet onto the department and the section field input values.

With one of the Dynamic Select Boxes selected in the *Design* view, select the *Details* tab. Notice that the weblet has adopted the field name and value:



2. In the *Design* view, select the DEPARTMENT dynamic select box weblet and use the *Details* tab to define its properties as follows:

Property	Value
listname	<b>DEPTDD</b>
codeField	<b>DEPARTMENT</b>
captionField	<b>DEPTDESC</b>

3. Select the Section dynamic select box weblet and define its properties as follows:

Property	Value
	<b>SECTDD</b>

Listname	
codeField	<b>SECTION</b>
captionField	<b>SECDESC</b>

**Note:** All properties can be selected from a dropdown list.

4. Adjust the width of dynamic select dropdowns, so that the department and section descriptions will be visible.
5. Save your changes and test your WAM by running the **Begin** web page.

At this stage:

Both the dropdown lists should be populated, the section's dropdown list containing only the sections for the current department.

When employee details are displayed, the dropdown lists should display the correct value for the employee. You could use your WAM `iiiEmpUpdate` to check this.

If you select a different section and update the employee, the update should be processed correctly.

The section's dropdown list is populated once only, so if you change department, it will contain an incorrect list of values. You will correct this in the next step.

### Step 3. Make the Sections Dropdown list dynamic

In this step you will define a new response WebRoutine which the AJAX enabled Dynamic Select box for sections will invoke.

1. Select the *Source* tab.
2. Create a new response WebRoutine, that will be invoked by the sections dynamic select box when DEPARTMENT changes. The requirements for this WebRoutine are:

The WebRoutine statement must have a Response() keyword with a value of SON

A web\_map for input, field DEPARTMENT

A web\_map for output of the list SectDD, defined as a \*JSON list

Invoke the buildDD2 method routine passing DEPARTMENT and SECTION

Your code should look like the following:

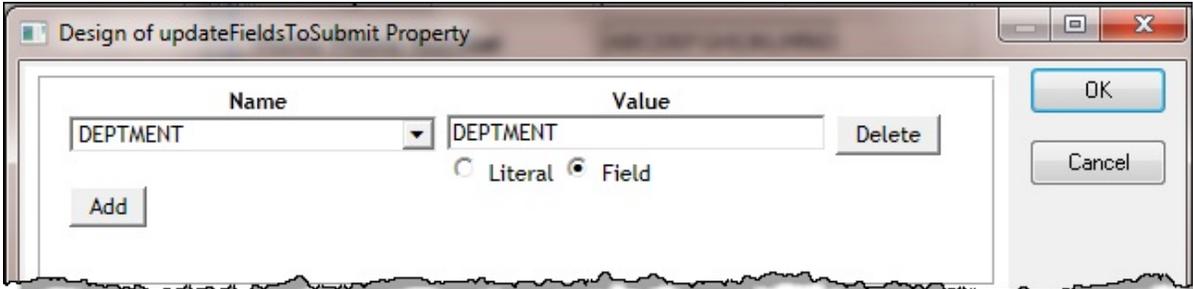
```
WebRoutine Name(USectDD) Response(*JSON)
Web_Map For(*input) Fields(#deptment)
Web_Map For(*output) Fields((#sectdd *JSON))
#com_owner.buildDD2 I_Dept(#deptment) I_Sect(#section)
Endroutine
```

3. Compile your WAM.
4. Open WebRoutine Details in the Design view.
5. Select the Section's dynamic select box and set up additional properties as follows:

```
updateOnFieldChange DEPARTMENT
updateWrName uSectDD
updateFieldsToSubmit Field: DEPARTMENT
Field Value: DEPARTMENT
```

6. Set the updateFieldsToSubmit property by selecting the *Value* column and clicking on the Ellipsis  button to open the *Design of...* property's dialog.

- a. Select a field *Name* of DEPARTMENT
- b. For *Value*, enter DEPARTMENT.
- c. Select the *Field* radio button.



7. Click *OK* to close the dialog.
8. Save your WAM.
9. Test your WAM by running the Begin webroutine. You should now be able to select a new department and notice that the sections dropdown list is refreshed.

In a later exercise, you will learn more about using the dynamic select box weblet.

## **Summary**

### **Important Observations**

- The Dynamic Select Box weblet is AJAX enabled. Its entries can be defined by a working list using a \*JSON attribute.
- A dynamic select box can dynamically refresh its list of values by calling a response WebRoutine when a monitored field value changes. The response WebRoutine must have a Response(\*JSON) keyword.

### **Tips & Techniques**

- Method routines can be used in a Web Application Module.

### **What You Should Now Know**

- How to implement dynamic select box weblets using a working list.
- How to setup a dynamic select box to refresh its list of values from the server.

## **WAM045 - A Dynamic Selector Dropdown list using a Select Field**

### **Objectives**

This example changes the implementation of the Dynamic Selector Dropdown list weblet for section code, by making use of a third field in the sections working list. The working list will now be defined as:

```
Def_List Name(#sectdd) Fields(#deptment #section #secdesc) Type(*Working
```

The list of sections will now be built once only and the selector field (DEPARTMENT) will enable the weblet to select the correct values to show, based on the value of field DEPARTMENT. A response WebRoutine for the Dynamic Selector for SECTIONS is no longer required.

The Dynamic Selector weblet for field SECTION will now to be set up to only display values that match the selector field, DEPARTMENT.

The list SECTDD will now contain all values from the table SECTAB. This technique works well if the total number of sections is small, as in this case. However, if the possible list of department codes and section codes is large (1,000's for example rather than 100's), then the solution implemented in exercise WAM040 will be a better solution. As usual there is a trade off to consider. Is it better to output all values of section to the web page once, or to refresh the list of sections every time the department code changes? Bear in mind that this second approach is itself efficient because it uses AJAX techniques to only refresh the sections list and not the whole web page.

To demonstrate this technique you will complete the following :

[Step 1. Create WAM iiiDynamSelector – Dynamic Selector using Select Field](#)

[Step 2. Setup the Dynamic Selector Dropdown list for Sections](#)

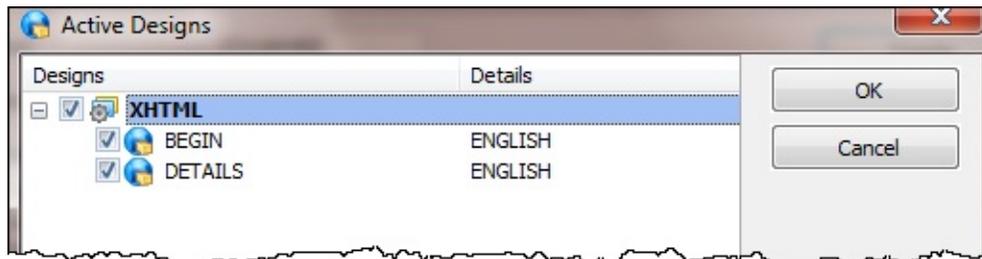
[Summary](#)

### **Before You Begin**

You should complete all preceding exercises.

## Step 1. Create WAM iiiDynamSelector – Dynamic Selector using Select Field

1. Create WAM iiiDynamSelector – Dynamic Selector using Select Field by copying WAM iiiEmpUpdate\_MK2 including the Active Designs:



2. Change the definition of the SECTDD list to include DEPARTMENT:  
Def\_List Name(#sectdd) Fields(**#department** #section #secdesc) Type(\*Working
3. Remove the USect\_DD WebRoutine, which is no longer required.
4. In the **Details** webroutine, move the code to invoke the BuildDD2 method routine to the position shown, at the end of the When (= M) logic. The moved code is shown in red. This routine will now be invoked once only, when the Details WebRoutine is initially called from the Begin web page.

```
WebRoutine Name(Details) Desc('Employee Details')
Web_Map For(*BOTH) Fields(#EMPDATA (#deptdd *json) (#sectdd *json))
Case (#STDREENTRY)
When ('= M')
Fetch Fields(#EMPDATA) From_File(PSLMST) With_Key(#EMPNO) Val_Er
If_Status Is_Not(*OKAY)
Message Msgtxt('Employee not found')
Transfer Toroutine(Begin)
Endif
Change Field(#EMPNOW) To(#EMPNO)
```

**#com\_owner.BuildDD2 I\_Dept(#department) I\_Sect(#section)**

5. Change the BuildDD2 method routine to match the following:  
The Select now reads all records (no With\_key(#department) ).

An If / Endif block has been removed

The code shown in red has been added.

```
Mthroutine Name(BuildDD2)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Define_Map For(*input) Class(#section) Name(#i_sect)
Clr_List Named(#sectdd)
* Read all Section records
Select Fields(#sectdd) From_File(sectab)
Add_Entry To_List(#sectdd)
Endselect
Loc_Entry In_List(#sectdd) Where((#deptment = #i_dept) And (#section =
Endroutine
```

The LOC\_ENTRY statement sets the values of DEPARTMENT and SECTION to the values retrieved for the current employee.

6. Compile your WAM.

## Step 2. Setup the Dynamic Selector Dropdown list for Sections

1. Open the Details WebRoutine in the Design view.
2. Select the Dynamic Selector weblet for field SECTION and change its properties as shown:

<b>Property</b>	<b>Value</b>
listname	<b>SECTDD</b>
selectorField	<b>DEPARTMENT</b>
selectorValueField	<b>DEPARTMENT</b>
codeField	<b>SECTION</b>
captionField	<b>SECTDESC</b>
updateOnFieldChange	<b>DEPARTMENT</b>
updateWrName	
updateFieldsToSubmit	

**Note that the updateWrName and updateFieldToSubmit properties no longer have a value.**

3. Save your changes
4. Execute your WAM in the browser to test (run the begin WebRoutine).
  - a. Change department and section code values.
  - b. Redisplay the employee and ensure that the changes have been correctly processed.

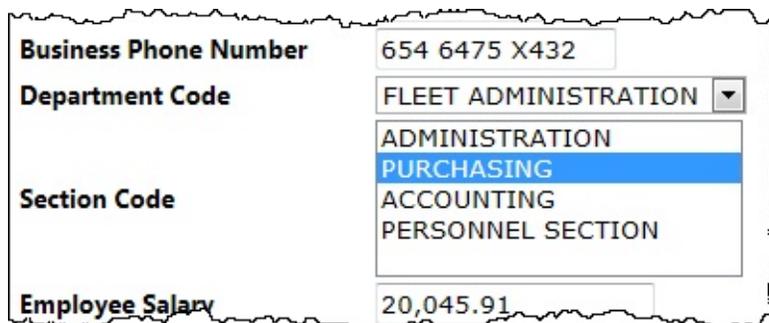
## Summary

### Important Observations

- As noted in the Objectives section, choosing the best technique to use requires a good understanding of your database and application.

### Tips & Techniques

- The Dynamic Select weblet may be used in a number of other ways. See the Web Application Modules guide for further details.
- For example, change the size property to 5, to display section codes as shown.



The image shows a screenshot of a web form with four fields. The 'Department Code' field is a dropdown menu with 'FLEET ADMINISTRATION' selected. The 'Section Code' field is a dropdown menu with 'PURCHASING' selected. The other fields are text boxes containing '654 6475 X432' for 'Business Phone Number' and '20,045.91' for 'Employee Salary'.

<b>Business Phone Number</b>	654 6475 X432
<b>Department Code</b>	FLEET ADMINISTRATION ▼
<b>Section Code</b>	ADMINISTRATION PURCHASING ACCOUNTING PERSONNEL SECTION
<b>Employee Salary</b>	20,045.91

- The weblet also supports multiple selections and return a list of fields to a WebRoutine.
- The updateFieldsToSubmit may be a list of field names, rather than a single value.

### What You Should Know

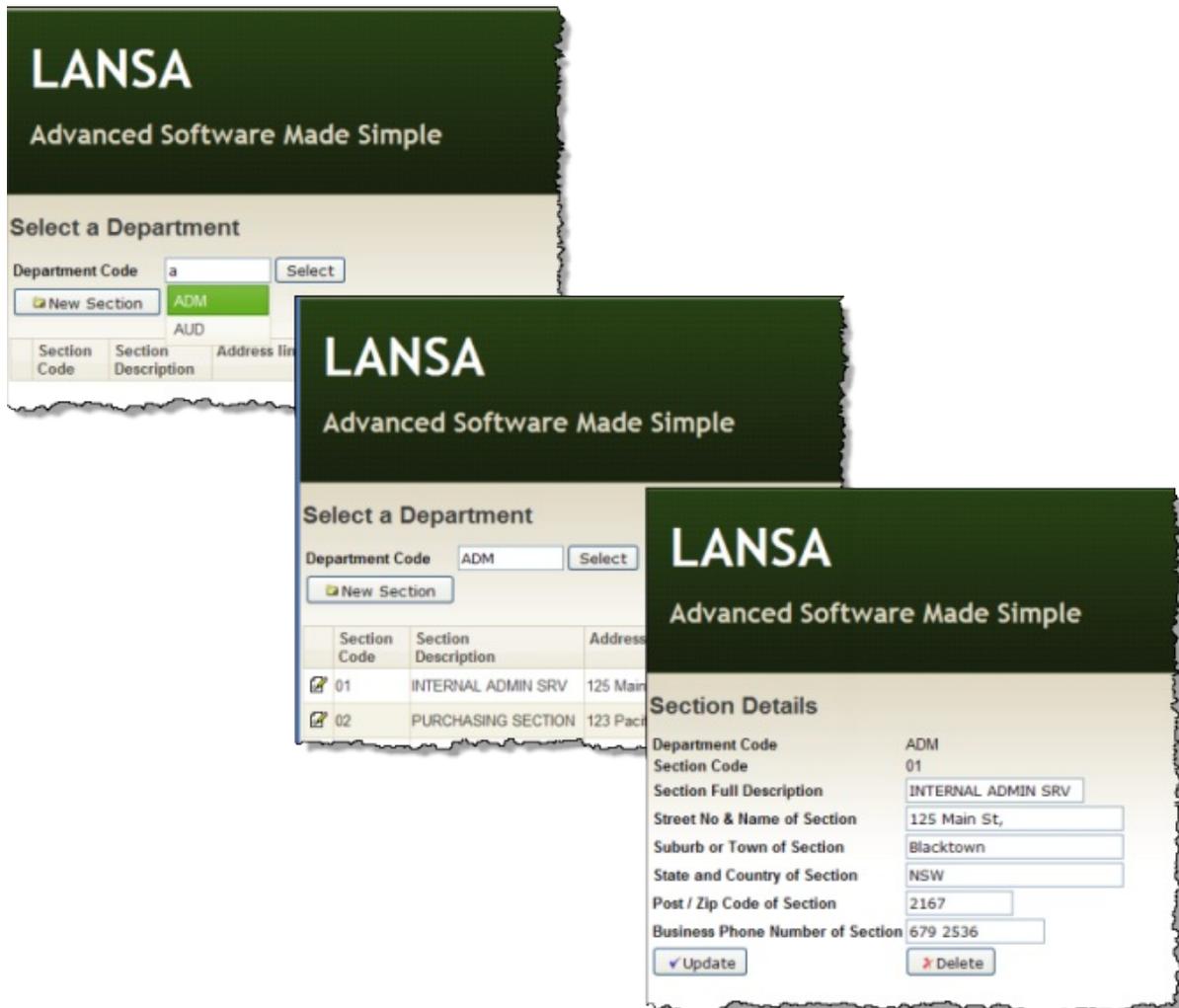
- You should now be aware that the Dynamic Selector weblet has a large number of features that may be used to enhance your user interface and the applications functionality.

# WAM050 - A Section Maintenance Application

## Objectives

- This exercise introduces an application that includes a list which is used to present a result set which enables selection of a single entry to maintain a record.
- To write a Section Maintenance WAM to display, update, delete and add sections. The main page will be based on a list of sections for the selected department.
- To show how lists can be used in a WebRoutine web page.
- To implement an example of the AutoComplete weblet.

Your completed application will look like the following:



To achieve these objectives you will complete the following steps:

Step 1. Create iiiSecMaint - Section Maintenance WAM

Step 2. Add a Details WebRoutine

Step 3. Create iiiSecAdd - Add Section WAM

Step 4. Complete the AddSect WebRoutine

Step 5. Set up the 'New Section' button on the Begin page for ii SecMainti

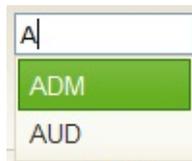
Summary

## Before You Begin

- In order to complete this exercise, you should have completed all the preceding exercises.

## The AutoComplete Weblet

The AutoComplete weblet provides suggestions while you type into the field. The suggestions are provided by a WebRoutine using Ajax.



A *sourceWrName* property defines a called WebRoutine, which supports the AutoComplete weblet by returning a list as JSON data. The WebRoutine must have a *Response(\*JSON)* keyword. The weblet calls this WebRoutine when you type into the input box.

A *listName*, *labelField* and *ValueField* define the list of values that the AutoComplete weblet displays.

Note: The AutoComplete weblet in this exercise will display matching department codes by reading the file DEPTAB using *Generic(\*yes)*. This provides a simple introduction to implementing this weblet, but does not provide a realistic example. Since the file DEPTAB has a very small number of records a complete list of values shown in a combo box would be a better implementation. You will also use the Dynamic List Box that was part of the previous exercise.

## Step 1. Create **iiiSecMaint** - Section Maintenance WAM

To design and develop your WAM applications, just like any other new application, you need to begin by focusing on:

- What is the business process?
- The user interface
- A new web application will require many WAMs and many web pages. Start by drawing up a plan of the web site and the navigation it will require.
- What WebRoutines will be required?
- What working lists, group\_bys, working fields need to be defined?
- What weblets can be used to enhance the user interface?
- What working lists will be required to support these weblets?
- How do fields and lists need to be mapped into and out of each WebRoutine?
- In a new WAM create your WebRoutines and WEB\_MAPS before coding your application logic
- When you add a new WebRoutine to a WAM, the XML/XSL will be built at compile time.

Avoid creating very large WAMs with too many WebRoutines. Each time a WebRoutine is invoked from the browser, the WAM loads and then unloads. Many small WAMs is a much better design and will be easier to maintain.

1. Create a new WAM.

Name: **iiiSecMaint**

Description: **Section Maintenance**

Layout weblet: **iiilay01**

Consider the first two screen captures shown in the *Objectives* section that show the **Begin** WebRoutine in operation. This WebRoutine initially displays a list of sections that is empty and an input field for a department code. An AutoComplete weblet will replace the Department Code input field. A *Select* push button invokes the **Begin** WebRoutine that builds a list of sections for the department and displays this list.

Consider what working lists will be needed to handle this web page and what logic will be necessary?

- a. Define a work field DEPT\_IN based on DEPARTMENT, this will be the department code input field.
- b. Define working list DEPTS, to support the AutoComplete weblet. The list contains the field DEPARTMENT only.
- c. Define working list SECTLIST, to support the list of sections containing STDSELECT, SECTION, SECDESC, SECADDR1. Note: All fields apart from STDSELECT should have an \*output attribute.
- d. Map STDREENTRY globally as a hidden field.

2. Define a WebRoutine named *Begin*, based on the following pseudo code

Map field DEPT\_IN for \* bothMap for \*output the Sections working list, SECTLISTCase of field STDREENTRY\* Select Push ButtonWhen = Sclear list SECTLISTselect fields in working list from the section table with the key dept\_inadd entry to working listend of select loopend of case loop

3. Define a response WebRoutine AutoComplete to build the list DEPTS to support the AutoComplete weblet

Define WebRoutine AutoComplete with a Response(\*JSON)  
keywordMap DEPT\_IN for inputMap list DEPTS for output as a \*JSON listConvert the first character of DEPT\_IN to uppercaseclear list of departmentsSelect DEPARTMENT from the file DEPTAB with the key DEPT\_IN, with a Generic(\*yes) keywordAdd an entry to the department listend selectend subroutine

Your completed code should now look like the following:

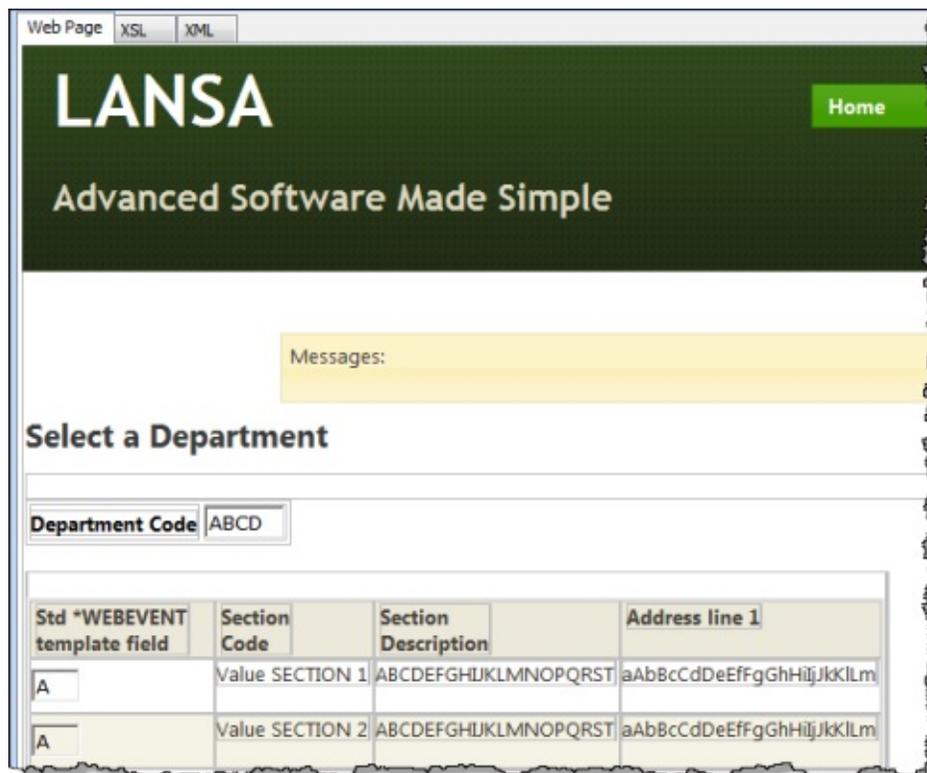
```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iilay01')
Define Field(#dept_in) Reffld(#department)
Def_List Name(#depts) Fields(#department) Type(*Working)
Def_List Name(#sectlist) Fields(#STDSELECT (#SECTION *out) (#SECDES
Web_Map For(*both) Fields((#stdreentry *hidden))
WebRoutine Name(Begin) Desc('Select a Department')
Web_Map For(*both) Fields(#dept_in #sectlist)
Case (#stdreentry)
```

```

When (= S)
Clr_List Named(#sectlist)
Select Fields(#sectlist) From_File(sectab) With_Key(#dept_in)
Add_Entry To_List(#sectlist)
Endselect
Endcase
Endroutine
WebRoutine Name(AutoComplete) Response(*JSON)
Web_Map For(*input) Fields(#dept_in)
Web_Map For(*output) Fields((#depts *json))
#dept_in := #dept_in.substring( 1, 1 ).upperCase
Clr_List Named(#depts)
Select Fields(#deptment) From_File(deptab) With_Key(#dept_in) Generic(*ye
Add_Entry To_List(#depts)
Endselect
Endroutine
End_Com

```

4. Compile your WAM and open the Begin WebRoutine in the Design view.  
Your web page should look like the following:



5. Drag and drop an jQuery UI AutoComplete weblet onto the Department Code field. Select the *Details* tab and set up the AutoComplete properties:

Property	Value
sourceWrName	<b>AutoComplete</b>
Listname	<b>DEPTS</b>
valueField	<b>DEPARTMENT</b>

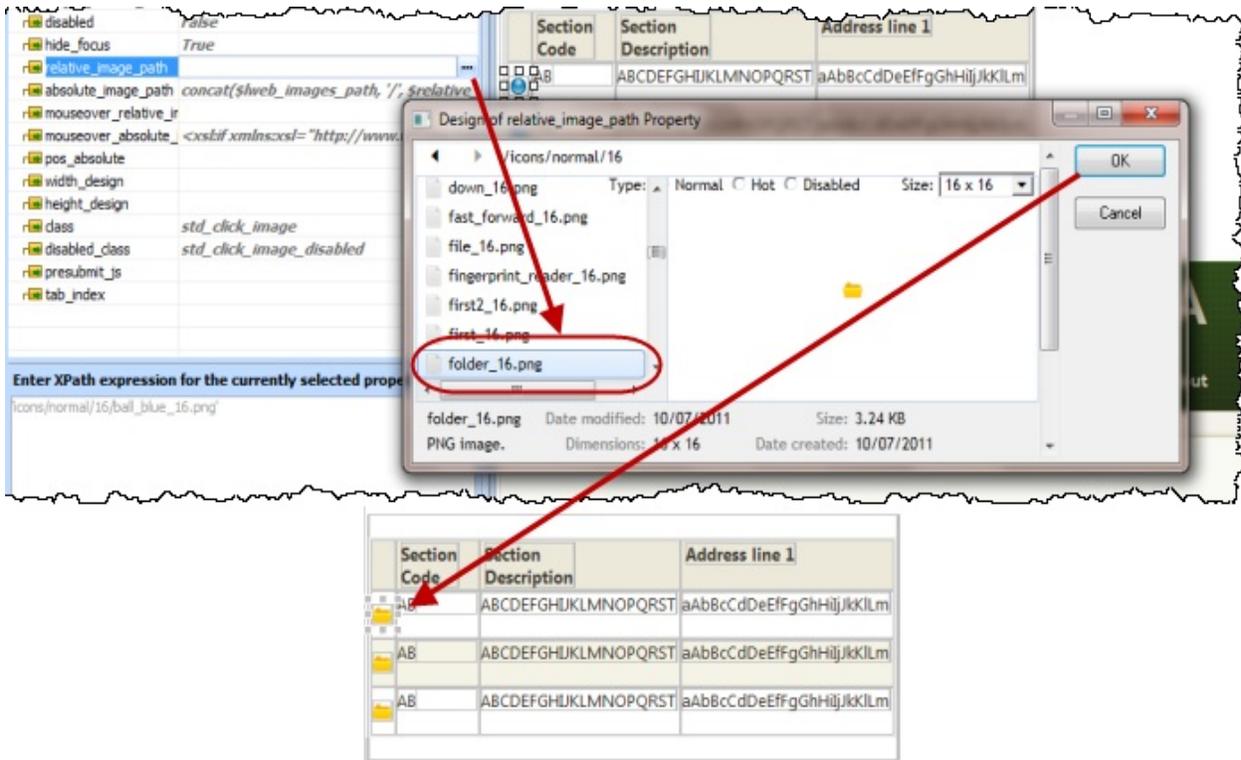
6. If the *Select* column (field STDSELECT) is shown as an input field (i.e. it does not have a clickable image weblet field visualization defined in the Repository), drop a Clickable Image weblet into the field in the first column.
7. With the clickable image selected, select the *Details* tab and set up its properties:

Property	Value
currentrowhfield	<b>SECTION</b>
currentrownumvalue	<b>\$SECTION</b>
Rentryvalue	<b>S</b>
tooltip	<b>Select a Section</b>
on_click_wrname	<b>Details</b>

**Note:**

- To return a field value from a list define the value as \$FIELDNAME (upper case). i.e. \$SECTION in this case.
- Although 'SECTION' is an \*output field in the list, you can return a field value via the clickable image weblet.
- The **Details** WebRoutine is not yet defined in your WAM, so you need to type in this value for the on\_click\_wrname property.

8. Select a clickable image weblet. Set the `relative_image_path` by clicking in the Value column, and then using the *Elipsis*  button and select the `/normal/16` folder and then select any suitable image. See the example following:



9. Select the column heading "Std \*WEBEVENT template field" and delete it.

10. If the field SECTION has a dropdown field visualization defined in the Repository, it will not be displayed in the list – since it is an \*output field. If necessary, select the field SECTION in the list and use the context menu to select the option *Replace with an Output Field*.

11. Select anywhere in the table containing "Department Code" and the AutoComplete weblet. Use the context menu, and select the option Table Items / Add columns... to add one column to the right. Add a *push button* into this new column and remove the \* placeholder characters from this cell.

Set up the push button properties:

Property	Value
caption	<b>Select</b>
on_click_wrname	<b>Begin</b>

submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: S</b>

**Note:** Whenever possible, set a weblet property value by selection from its dropdown list. Using a list of values provided by the editor, when available, will help to minimize errors in your XSL.

12. Save your changes and run the web page in the browser. Your page should look like the following:



## Step 2. Add a Details WebRoutine

1. This WebRoutine will support display, update and delete of the selected Section record. Consider the logic that needs to be supported by this WebRoutine.
  - a. What fields and lists need to be mapped?
  - b. How to control the code to be executed each time this WebRoutine is invoked?
  - c. What work fields will be required to retain key values?
  - d. Whether to transfer control back to the **Begin** WebRoutine after a successful update or deletion?
2. At the WAM level, define a work field and a Group\_by statement.
  - a. Define a work field, SECTW based on field SECTION. This will retain current value of SECTION.
  - b. Define a Group\_by, SEC\_DETL for all fields in the file SECTAB. Fields DEPARTMENT and SECTION should have a display attribute of \*output.
3. Create a **Details** WebRoutine based on the outline following:

Map for \*both the Group\_by for Section fields and fields DEPT\_IN and SECTW. Fields DEPT\_IN and SECTW should be hidden fields.

Use field STDREENTRY to determine why the WebRoutine was called.

When called initially, fetch the Section record and save department and section code in work fields.

When called for update, set up department and section code from work fields and update the record.

If the update is successful, output a message and transfer to the **Begin** WebRoutine. Consider how the **Begin** routine should redisplay Sections for current department.

Output a message if the update is not successful.

When called for delete, set up department and section code from work fields, delete the record, if successful, output a message and transfer to **Begin** WebRoutine.

Output a message if the delete is not successful.

Your complete code for the **Details** WebRoutine should look like the following:

```
WebRoutine Name(Details) Desc('Section Details')
Web_Map For(*BOTH) Fields(#SEC_DETL (#SECTW *HIDDEN) (#dept_in
Case Of_Field(#STDREENTRY)
* Initial call from clickable image
When Value_Is(= S)
Fetch Fields(#SEC_DETL) From_File(SECTAB) With_Key(#dept_in #SECTI
#dept_in := #DEPARTMENT
#SECTW := #SECTION
* Update button clicked
When Value_Is(= U)
* ensure that DEPARTMENT and SECTION can be redisplayed if a validation er
#DEPARTMENT := #dept_in
#SECTION := #SECTW
Update Fields(#SEC_DETL) In_File(SECTAB) With_Key(#dept_in #SECTW
If_Status Is(*OKAY)
#STDREENTRY := L
Message Msgtxt('Section changed')
Transfer Toroutine(BEGIN)
Else
Message Msgtxt('Error occurred on update')
Endif
* Delete button clicked
When Value_Is(= D)
Delete From_File(SECTAB) With_Key(#dept_in #SECTW) Val_Error(*NEXT
If_Status Is(*OKAY)
#STDREENTRY := S
Message Msgtxt('Section deleted')
Transfer Toroutine(BEGIN)
Else
Message Msgtxt('Error occurred on deletion')
Endif
Endcase
Endroutine
```

4. Recompile your WAM and open the *Design* view for the Details WebRoutine. Your page should look like the following:

### Section Details

Department Code	Value DEPARTMENT
Section Code	Value SECTION
Section Full Description	ABCDEFGHIJKLMNOPQR:
Street No & Name of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
Suburb or Town of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
State and Country of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code of Section	123456
Business Phone Number of Section	ABCDEFGHIJKLMN

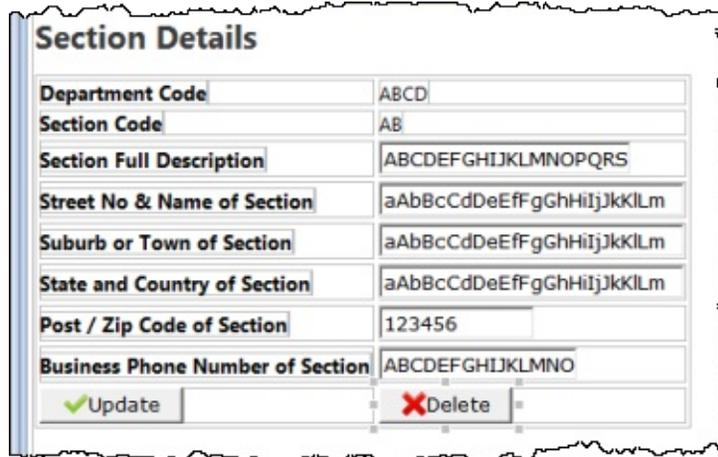
- If the Department and Section code fields have a combo box field visualization weblet defined in the *Repository*, select each of them and use the context menu to Replace with output field. Fields with a visualization weblet defined, will not display on the page in "output" mode.
- Use the context menu to Add a row to the bottom of the table, and drop a Push Button with Image into each cell.
- Set up the push button properties based on the following:

Property	Value
caption	<b>Update</b>
left_relative_image	<b>icons/normal/16/check_mark_16.png</b>
on_click_wname	<b>Details</b>
submitExtraFields	<b>Field Name: STDREENTRY</b> <b>Literal Value: U</b>
caption	<b>Delete</b>
left_relative_image	<b>icons/normal/16/cross_16.png</b>
on_click_wname	<b>Details</b>
submitExtraFields	<b>Field Name: STDREENTRY</b> <b>Literal Value: D</b>

The `left_relative_image` and `submitExtraFields` properties should be selected using the *Ellipsis* button and the *Design of...* dialog.

8. Save your web page design.

Your completed design should look like the following:



The image shows a web form titled "Section Details" with a torn-paper border. It contains several input fields with text, and two buttons at the bottom: "Update" with a green checkmark and "Delete" with a red X.

Field Label	Value
Department Code	ABCD
Section Code	AB
Section Full Description	ABCDEFGHIJKLMNQRS
Street No & Name of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
Suburb or Town of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
State and Country of Section	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code of Section	123456
Business Phone Number of Section	ABCDEFGHIJKLMNO

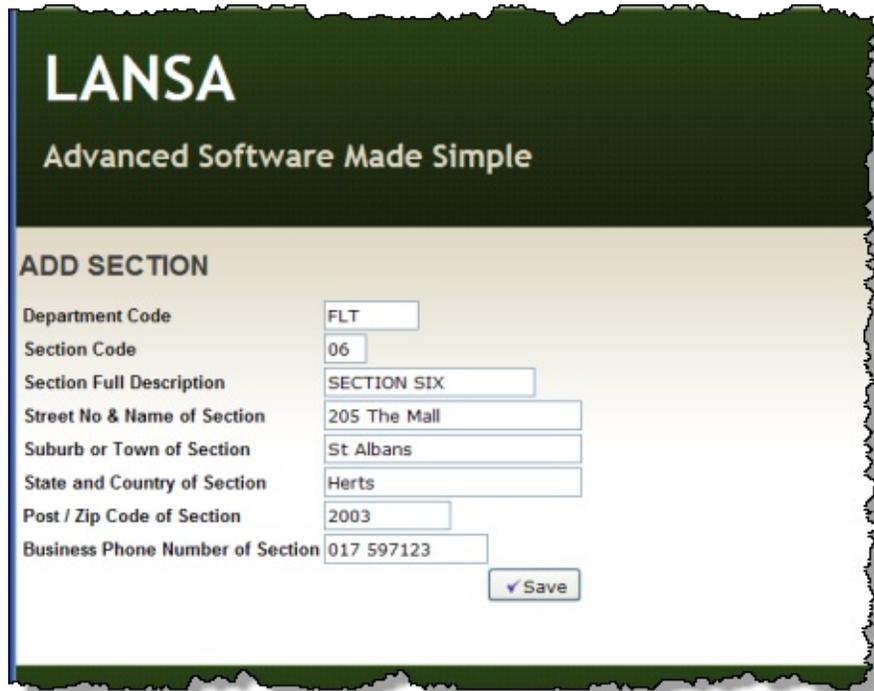
Buttons:

9. Re-test your WAM. Check what happens on a successful update or delete. Check what happens after an update with a validation error.

### Step 3. Create iiiSecAdd - Add Section WAM

This simple logic could be incorporated into WAM iiiSecMaint, however, you will instead create a new WAM to illustrate building a multi-WAM application.

1. Review the following web page:



The screenshot shows a web form titled 'ADD SECTION' from LANSA. The form has a dark green header with the LANSA logo and the tagline 'Advanced Software Made Simple'. Below the header, the form fields are as follows:

Department Code	FLT
Section Code	06
Section Full Description	SECTION SIX
Street No & Name of Section	205 The Mall
Suburb or Town of Section	St Albans
State and Country of Section	Herts
Post / Zip Code of Section	2003
Business Phone Number of Section	017 597123

At the bottom right of the form is a 'Save' button with a checkmark icon.

The AutoComplete weblet will support the department code input field

All other fields will be entered

A **New** button will be added to the **Begin** page for iiiSecMaint to call iiiSecAdd.

The Add Section WAM will return to Section Maintenance **Begin** WebRoutine when a section is successfully added.

Section Maintenance **Begin** WebRoutine will then list the sections for the department for which a section was added.

2. Create a new WAM **iiiSecAdd– Add Section** using *Layout Weblet iiilay01*. Consider what common definitions and logic you could copy from iiiSecMaint. What WEB\_MAPs will be required in the 'AddSect' WebRoutine?
3. Create a basic **outline** for your WAM based on the following pseudo code  
Define a work field DEPT\_IN based on DEPARTMENT  
Define a

Group\_by of all fields for the Section table. All fields should be input capable  
Map field STDREENTRY for \*both, as a \*hidden field  
Define a WebRoutine "AddSect"

Map field DEPT\_IN for \*both  
Map group\_by for Section fields for \*both, all fields should be input capable. DEPARTMENT should be a hidden field  
End routine

Your code should now look like this:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iilay01')
Define #DEPT_IN Reffld(#DEPARTMENT)
Group_By Name(#SECT_DETL) Fields((#DEPARTMENT *hidden) #SECTION
*)
Web_Map For(*BOTH) Fields((#STDREENTRY *HIDDEN))
*)
WebRoutine Name(AddSect) Desc('Add Section')
Web_Map For(*BOTH) Fields(#Dept_In #SECT_DETL)
* -----
* Add section logic goes here
* -----
Endroutine
End_Com
```

## Step 4. Complete the AddSect WebRoutine

1. Your **AddSect** WebRoutine will handle the initial call from the **Begin** web page in iiiSecMaint and the call from the Save button' on the **AddSect** web page itself.

Create your logic for WebRoutine **AddSect** based on the following pseudo code:

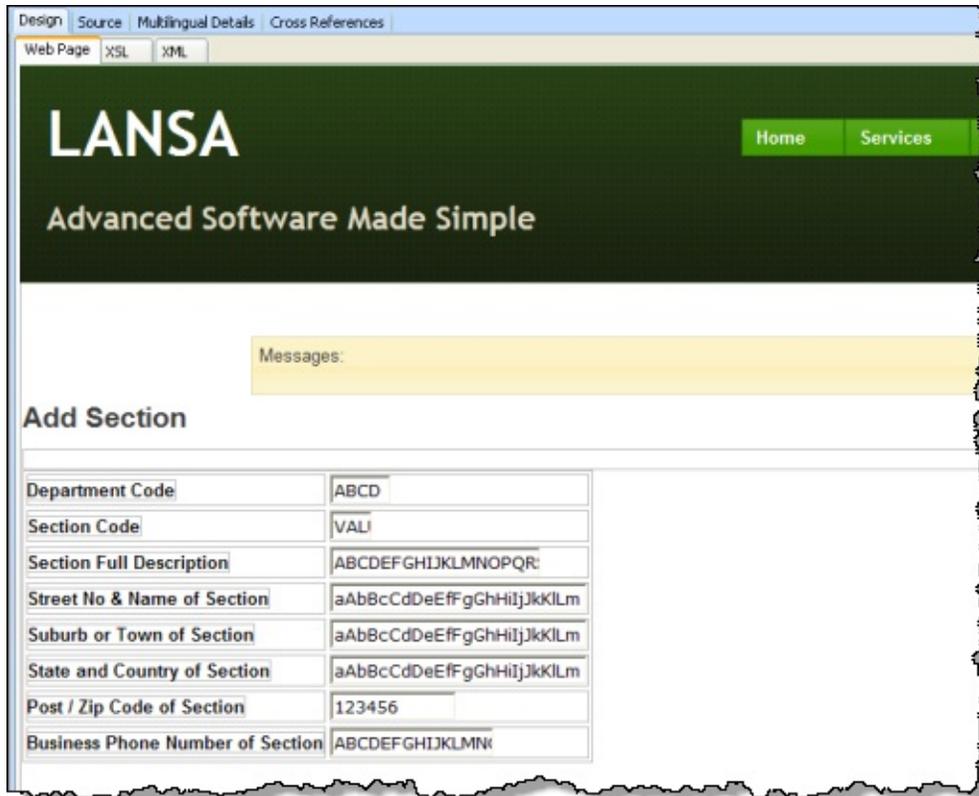
```
CASE of STDREENTRY* when called from Begin WebRoutineWhen
= NChange DEPARTMENT to DEPT_IN (value passed in from
iiiSecMaint)Change SECTION to *null (value is passed in from
iiiSecMaint)* when Save button clicked When = AChange
DEPARTMENT to DEPT_IN (the input value on the web page)Insert to
Section fileIf Status is *OKAYOutput messageChange STDREENTRY
to LTransfer to WebRoutine iiiSecMaint.BeginEndifEndcase
```

Your completed RDMLX code should look like the following:

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
Group_By Name(#SECT_DETL) Fields((#DEPARTMENT *hidden) #SECTION
Define Field(#dept_in) Reffld(#deptment)
Web_Map For(*BOTH) Fields((#STDREENTRY *HIDDEN))
WebRoutine Name(ADDSECT) Desc('ADD SECTION')
Web_Map For(*BOTH) Fields(#dept_in #SECT_DETL)
Case Of_Field(#STDREENTRY)
* New button clicked in iiiSecMaint
When (= N)
#deptment := #dept_in
#SECTION := *NULL
* Add button clicked
When (= A)
#deptment := #dept_in
Insert Fields(#SECT_DETL) To_File(SECTAB) Val_Error(*NEXT)
#STDREENTRY := *NULL
If_Status Is(*OKAY)
Message Msgtxt('Section successfully added')
#stdreentry := L
Transfer Toroutine(#iiiSecMaint.BEGIN)
```

Endif  
 Endcase  
 Endroutine  
 End\_Com

2. Compile your WAM and open in the Design view. It should look like the following:



3. If necessary change the Department Code to an input field. Drop an autoComplete weblet onto the department field value. Set up the weblet properties based on:

Property	Value
sourceWamName	<b>iiiSecMaint</b>
sourcewrName	<b>AutoComplete</b>
listName	<b>DEPTS</b>
valueField	<b>DEPARTMENT</b>

Note that the AutoComplete weblet is calling the response WebRoutine you created in WAM **iiiSecMaint**.

4. The Section field should be an input field, change it if necessary.
5. Add a row to the bottom of the table and add a push button with image to the right hand cell. Make the cell right align. Set up the push button properties as shown:

Property	Value
caption	<b>Save</b>
left_relative_image	<b>icons/normal/16/check_mark_16.png</b>
on_click_wrname	<b>AddSect</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: A</b>

6. Save these changes. Your completed AddSect page should look like the following:

**Add Section**

Department Code	ABCD
Section Code	VALI
Section Full Description	ABCDEFGHIJKLMNOPQR
Street No & Name of Section	aAbBcCdDeEfGhHiIjJkKlLm
Suburb or Town of Section	aAbBcCdDeEfGhHiIjJkKlLm
State and Country of Section	aAbBcCdDeEfGhHiIjJkKlLm
Post / Zip Code of Section	123456
Business Phone Number of Section	ABCDEFGHIJKLMN
	<input type="checkbox"/> <input checked="" type="checkbox"/> Save <input type="checkbox"/>

## Step 5. Set up the 'New Section' button on the Begin page for ii SecMainti

1. Open the Begin WebRoutine in WAM iiiSecMaint in the *Design* view.
  - a. Add a new row to the table containing the Department AutoComplete weblet.
  - b. Set the left hand cell in the new row, to align left.
  - c. Add a button with image to the bottom left hand cell.

**Hint:** If you have a problem selecting inside the table, select the push button and move the cursor right, to position in the table cell. Enter two \* characters. Now right click and use *Table Items / Add Rows...*
  - d. Delete the place holder characters.
2. Set up the New Section push button properties as shown:

<b>Property</b>	<b>Value</b>
Caption	<b>New Section</b>
left_relative_image	<b>icons/normal/16/star_16.png</b>
On_click_WamName	<b>iiiSecAdd</b>
On_click_wrname	<b>AddSect</b>
submitExtraFields	<b>Field Name: STDREENTRY</b> <b>Field Value: N</b>

- a. Adjust the width of the push button to display the caption as one line.
- b. Save your changes.

Your completed **Begin** page should look like the following:

### Select a Department

Department Code:

Section Code	Section Description	Address line 1
<input type="text" value="AB"/>	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfGhHiIjJkKlLm
<input type="text" value="AB"/>	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfGhHiIjJkKlLm
<input type="text" value="AB"/>	ABCDEFGHIJKLMNQRST	aAbBcCdDeEfGhHiIjJkKlLm

- Save these changes and run the **Begin** WebRoutine for WAM iiiSecMaint to test iiiSecAdd.
  - Adding a new section should return to the *Select a Department* web page and display a list of sections for the department concerned.
  - Trying to add a section with invalid or missing data should redisplay the *Add Section* web page, with error messages.

## Summary

### Important Observations

- A working list is displayed on the web page as a table
- Columns in the list may be input capable or output only.
- Columns in the list table may have weblets applied to them.
- Like fields, lists may be mapped for \*output, \*input or \*both.
- The clickable image weblet can return a current row field and value and a STDREENTRY value to the WebRoutine it calls.
- A second WAM can be invoked simply by specifying the on\_click\_wamname property.
- The AutoComplete weblet can be implemented to invoke a response WebRoutine in an existing WAM, as used in exercise [WAM050 - A Section Maintenance Application](#).

### Tips & Techniques

- Display, update and delete logic can easily be included in a single WebRoutine by using a CASE loop for field STDREENTRY
- Note that anchor weblets can also be used on fields in a browselist, to select a row – see later exercise.

### What You Should Now Know

- How working lists are handled in the web page.
- How to select an entry from a browselist.
- How to design an application with a number of WAMs.

## **WAM055 - Using LANSAs Debug**

### **Objectives**

To learn how to use interactive debug with WAMs.

Initially this exercise assumes you are running your WAMs locally. A later step will demonstrate running debug as the WAM executes on the IBM i server.

To achieve these objectives you will complete the following:

[Step 1. Get Started with Debug](#)

[Step 2. Use Breakpoints](#)

[Step 3. Use Break on Value Condition](#)

[Step 4. Use Debug when the WAM is running on the server](#)

[Summary](#)

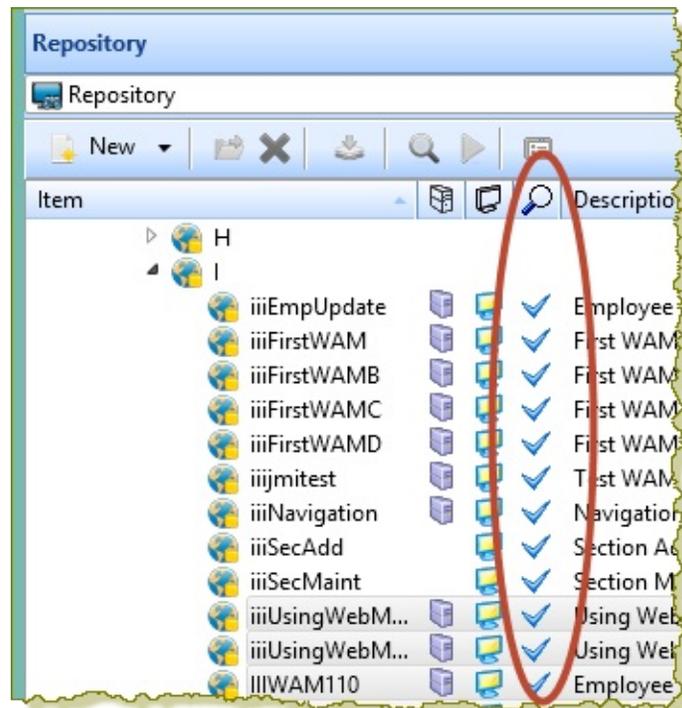
### **Before You Begin**

- In order to complete this exercise, you must have completed exercise WAM050.

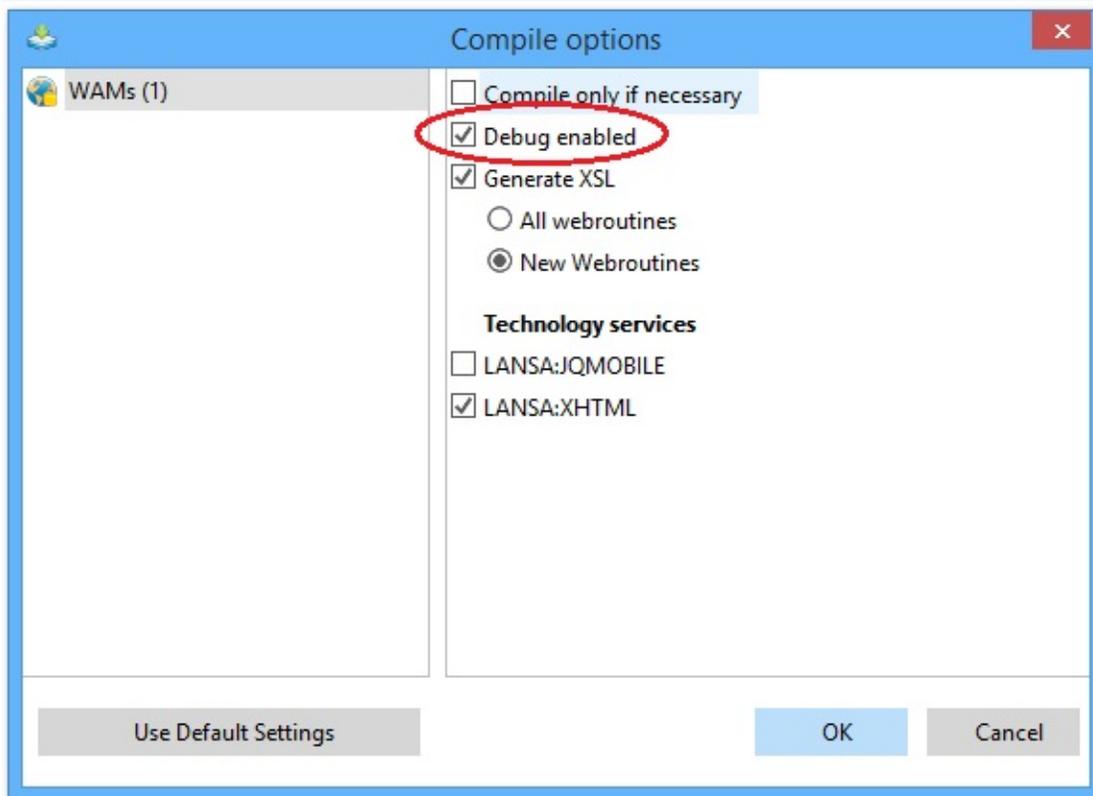
## Step 1. Get Started with Debug

This exercise uses **iiiSecMaint - Section Maintenance**, which must be compiled with debug enabled.

**Note:** You can check if your WAM is debug enabled in the *Repository* tab or *Last Opened* tab in the LANSa Editor.

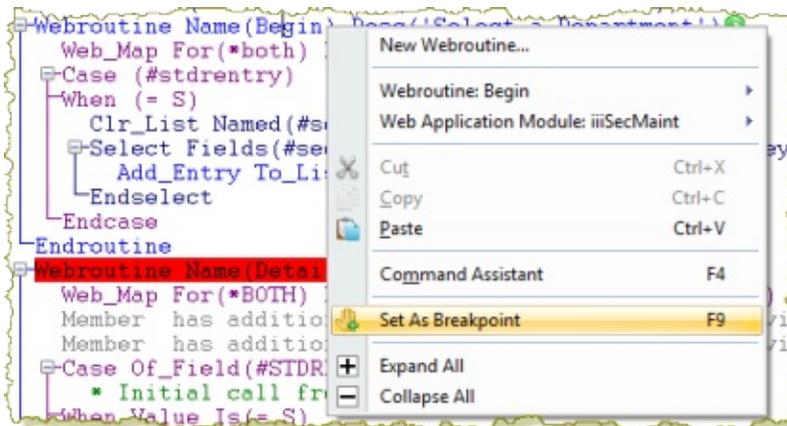


If necessary use the *Verify / Compile* menu option to recompile your WAM debug enabled.



## Step 2. Use Breakpoints

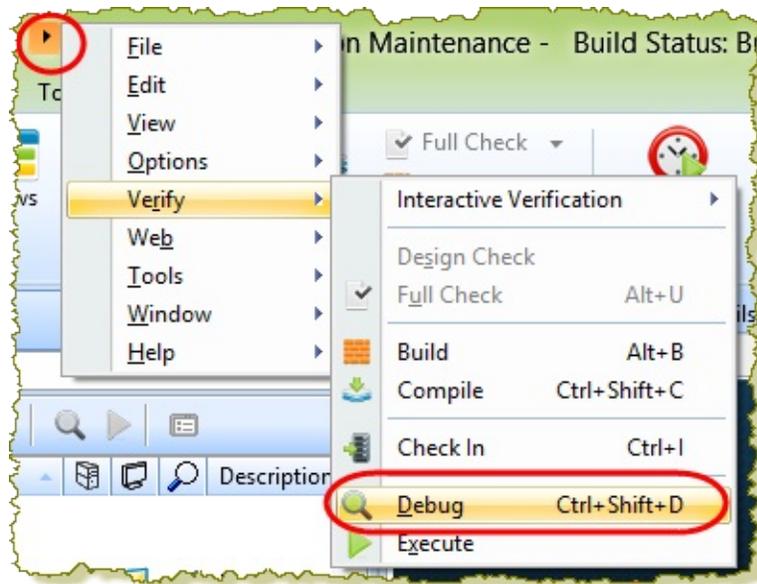
1. Open WAM iiiSecMaint in the editor. Set a breakpoint on the **WebRoutine name(Begin)** statement. To set the breakpoint, select the line and then use **F9** or use the context mouse menu option *Set As Breakpoint* option. Breakpoint lines are highlighted in red.



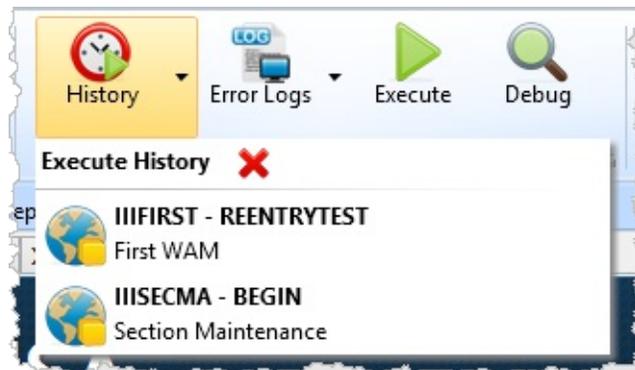
Your source code should now look like the following:



2. Open the WebRoutine **Begin** in *Design*. Run the WAM in debug mode using the  **Debug** *Run in Debug* button on the *Home* ribbon, or use the *Verify / Debug* menu option from the *Toolbar Menu* button:



**Note:** You can use the *History* button on the Home ribbon, to run a WAM/WebRoutine executed earlier.

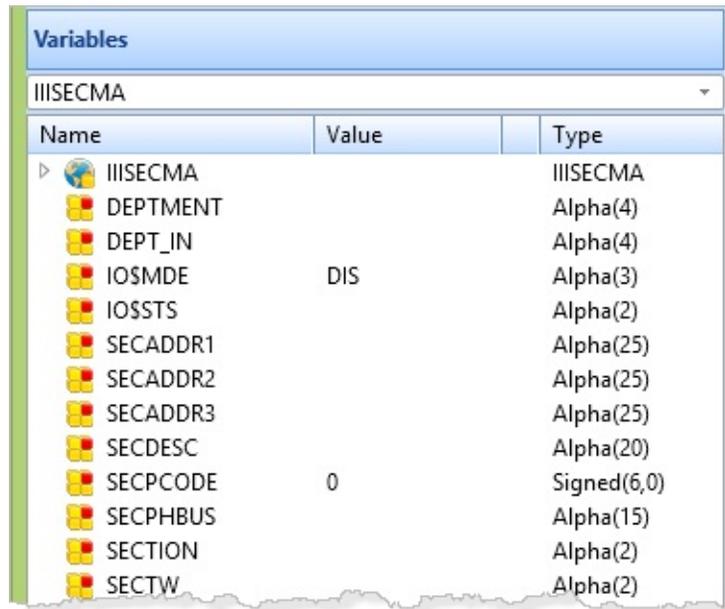


- When the WAM starts in debug mode in the browser, it will immediately switch focus to the LANSASoft Editor, because WebRoutine **Begin** is run immediately, and it has a breakpoint defined, the breakpoint WebRoutine statement will be highlighted in yellow. This is the statement that is about to be executed as shown:

```

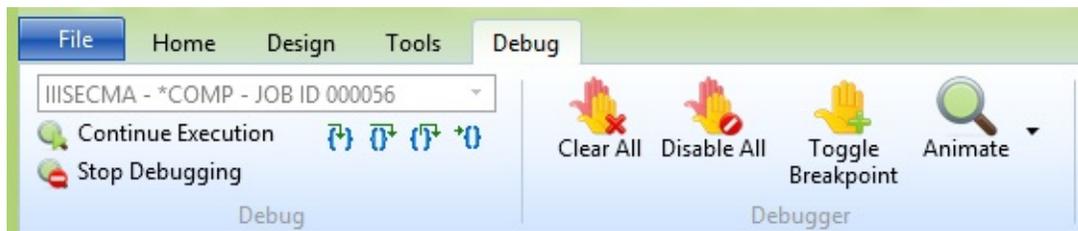
Web_Map For(*both) Fields((#stdreentry *hidden))
Webroutine Name(Begin) Desc('Select a Department')
Web_Map For(*both) Fields(#dept_in #sectlist)
Case (#stdreentry)
  When (= S)
    Clr_List Named(#sectlist)
    Select Fields(#sectlist) From_File(sectab) With_Key(#dept_in)
      Add_Entry To_List(#sectlist)
    Endselect
  Endcase
Endroutine
  
```

Note: in debug, program variables are shown on the *Variables* tab, in the left hand side pane, including working lists.

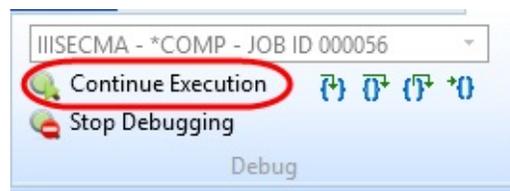


Name	Value	Type
IIISECMA		IIISECMA
DEPARTMENT		Alpha(4)
DEPT_IN		Alpha(4)
IOSMDE	DIS	Alpha(3)
IOSSTS		Alpha(2)
SECADDR1		Alpha(25)
SECADDR2		Alpha(25)
SECADDR3		Alpha(25)
SECDESC		Alpha(20)
SECPCODE	0	Signed(6,0)
SECPHBUS		Alpha(15)
SECTION		Alpha(2)
SECTW		Alpha(2)

A *Debug* ribbon is shown at the top of the editor:



- Use the *Continue Execution* button or F5, to run straight through this WebRoutine.

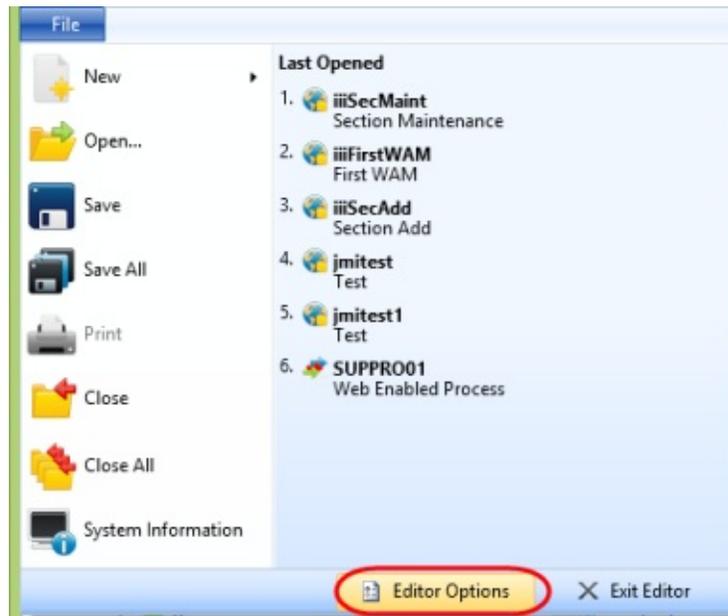


The initial page will be displayed in the web browser.

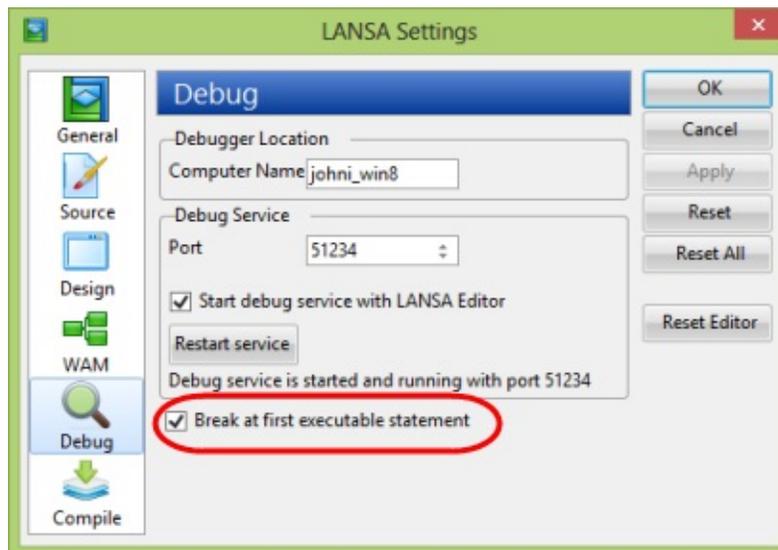
- Start typing into the department code input box and notice that debug stops at the AutoComplete WebRoutine:

```
Webroutine Name(AutoComplete) Response(*JSON)
Web_Map For(*input) Fields(#dept_in)
Web_Map For(*output) Fields((#depts *json))
#dept_in := #dept_in.substring( 1, 1 ).upperCase
Clr_List Named(#depts)
Select Fields(#deptment) From_File(deptab) With_Key(#dept_in) Get
Add_Entry To_List(#depts)
Endselect
Endroutine
```

This happened because Debug has a general setting to *Break at first executable statement*. To check this, select *Editor Settings* on the *File* menu:



c. Select the Debug icon.



- d. Uncheck the *Break at first executable statement* setting and click OK. Press F5 to continue running the WAM.
- e. With a valid department such as ADM or AUD entered, click the *Select* button. Focus will return to the WebRoutine statement for Begin in the editor and it will be highlighted in yellow as before.
- f.  Use the *Step Into* button or F8, to step through the WebRoutine one line at a time. If necessary use F5 to run through to the end of the Begin WebRoutine.
- g.  Use the *Run to Cursor* button to run all code up to the cursor position. To try this, press F5 to run through to the end of the WebRoutine. Enter a department code and click the *Select* button. In the Editor, position the cursor on the following statement in the Begin WebRoutine. Click the *Run to Cursor* button. You should run all code up to the SELECT statement and then stop.

SELECT FIELDS(#sec\_list) FROM\_FILE(sectab) WITH\_KEY(#deptment

- h.  Use the *Step Out* button to run the code up to next breakpoint. In the editor set the SELECT statement to be a breakpoint and run the Begin WebRoutine again. Now use the *Step Out* button to execute the routine up to the SELECT statement. You could also have used the F5 key to achieve this. Use F8 to run around the SELECT loop. Notice the changing values of the variables shown in the *Variable* tab. That is, the value of variables DEPARTMENT, SECTION, SECDESC etc. If necessary now use F5 to run to the end of the WebRoutine.
- i.  Use the *Step Over* button to execute the procedure called by the current line and break at the line following the current line. *Step Over* is identical to *Step Into* except when the current statement contains a call to a procedure, *Step Over* executes this procedure as a unit and then steps to the next statement in the current procedure.

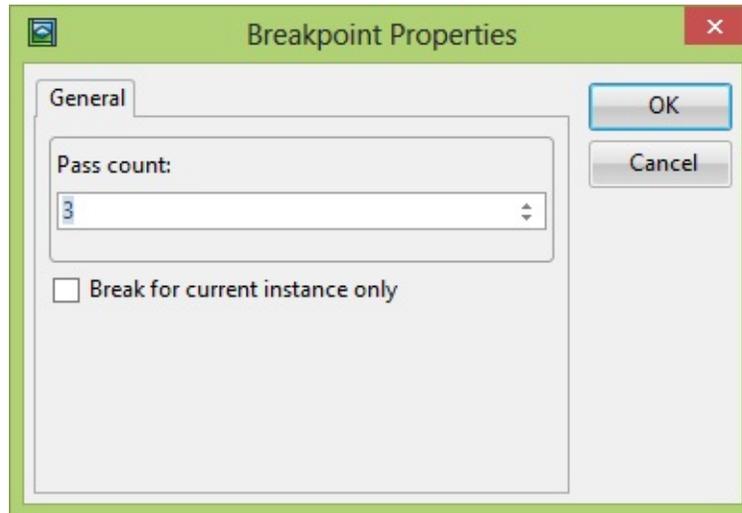


- j. Use the *Toggle Breakpoint* button on the *Debug* ribbon. In the web browser, enter a department code and click the *Select* button. In the LANSAs Editor, clear the breakpoint on the SELECT statement. To do this, select the SELECT line, and use the F9 key, or the *Toggle Breakpoint* button on the *Debug* ribbon to clear this breakpoint.

4. Set a breakpoint on the line

`ADD_ENTRY TO_LIST(#sec_list).`

Use the right mouse menu and select  *Breakpoint Properties*, and set the pass count to 3 and press OK.

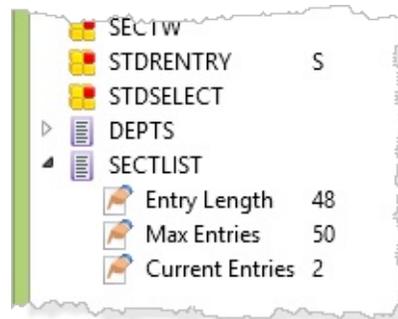


5. Remove the breakpoint from the Begin WebRoutine statement.

a. Press **F5** twice to run to the end of the WebRoutine.

b. When focus returns to the LANSAP Editor, expand the **SECT\_LIST** entry in the *Variables* tab.

**Note:** The list SECT\_LIST contain 2 entries currently.



Also notice the *Breakpoint* tab shows a passcount of 3 for this breakpoint. i.e. the `ADD_ENTRY` statement is about to be executed for the third time.

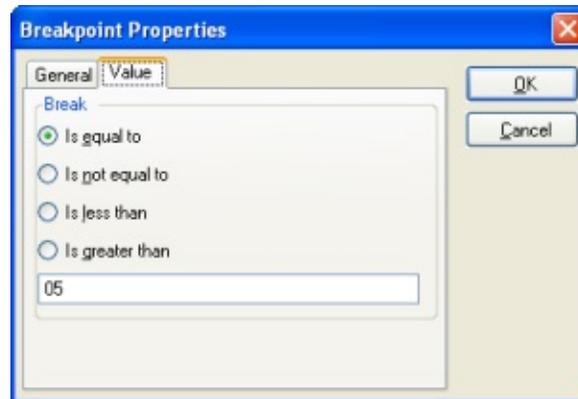
Breakpoints	Pass Count	Hit Count	Line Number	Statement Text
IIISECMA	0	1	14	Webroutine Name(E
IIISECMA	3	3	20	Add_Entry To_List(#

If the *Breakpoints* tab is not visible, select it from the *Home* ribbon /



*Views* gallery.

- d. Press **F5** to continue running. Note that each time the program breaks, 2 more entries have been added to the list (the third entry is about to be added).



5. Click OK to save the condition and press F5 to continue running the WAM. The breakpoint will next occur when SECTION has the chosen value.

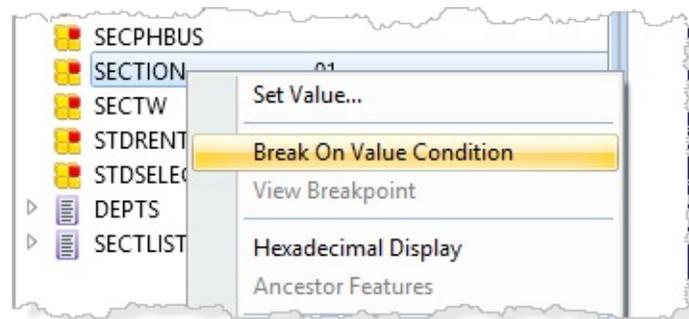
### Step 3. Use Break on Value Condition

A breakpoint may have a variable value associated with it, meaning that that debug will break at this statement only when the value condition is true. To set a break on value condition you must first run the WAM in debug mode with a breakpoint set for the statement required. While running in debug. Select the *Variable* tab, select a variable and define a *Break on value* condition for this variable. This condition will be saved until it is removed. Continue running the WAM in debug mode and this statement will now break only when the variable's *Break on value condition* is true.

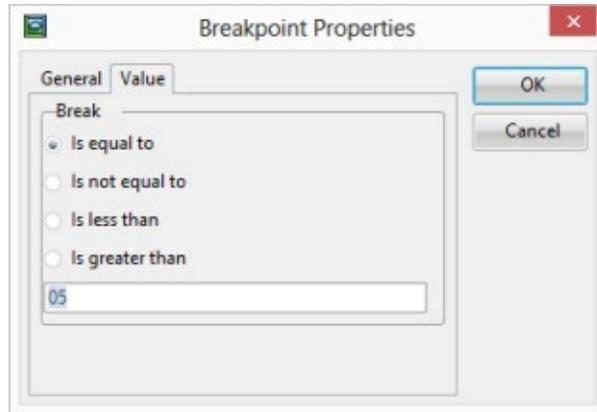
At this point your WAM iiiSecMaint should have a breakpoint defined on the statement:

Add\_Entry To\_List(#sectlist)

1. First run the WAM normally (not in debug mode) and review the list of section codes for the department you are using, for example ADM. Decide which section code value you want to break on, for example SECTION = 05.
2. Run the WAM again, this time in debug mode, enter a department code and click the Select button.
3. When debug breaks on the ADD\_ENTRY statement select the field SECTION on the Variables tab and use the context menu to set the Break on Value condition:

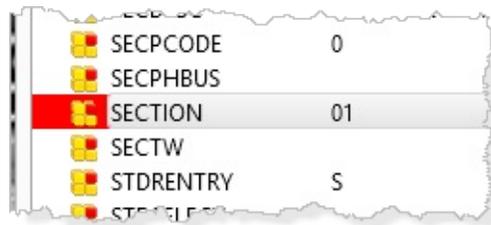


4. Set the condition to SECTION = 05 (or your chosen value).



5. Click OK to save the condition and press F5 to continue running the WAM. The breakpoint will next occur when SECTION has the chosen value.

The variable is highlighted on the *Variables* tab to show it has a *Break on Value* setting.



## Step 4. Use Debug when the WAM is running on the server

To complete this step you must be using a Slave Workstation installation of Visual LANSAs with a Master Repository on an IBM i server.

Debug with the WAM running on the server, requires communication from the developer's PC to the IBM i server and from the IBM i server to the developer's PC.

Visual LANSAs Debug is a service that is started when VL starts. See *File / Editor Options* and select *Debug*:



This shows that the debug service is running on Port 51234.

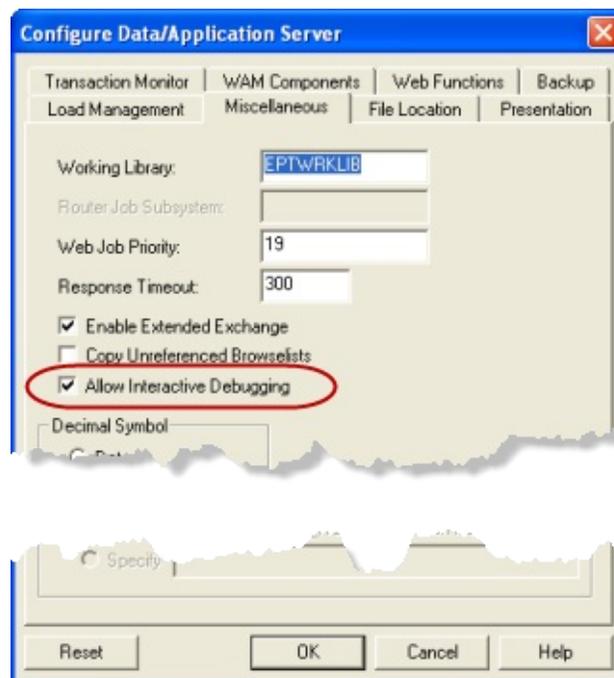
The IBM i server's communication to the developer's PC may rely on a locally defined Domain Name Server (DNS) to resolve the developer's PC Name to an IP address. Alternatively a routing entry must be defined for each developer's PC including the IP Address for the PC:

```
LCOADMP407      Change Communications Routing Record

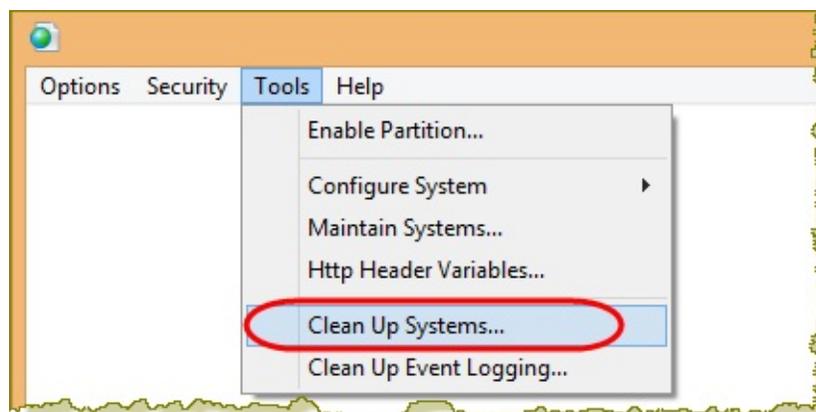
Partner LU name (Host) . . . . . A7800511
Fully qualified name . . . . . 10.44.10.51
-----
Communications method . . . . . SOCKET
Communications module . . . . .
Connection identifier . . . . . 4548
Packet size . . . . . 00000
TCP_NODELAY TCP/IP socket option . YES
Enable IPv6 . . . . . NO
```

To access this screen use the LANSA/CONFIGURE IBM i command and select, COMMS\_EXTENSIONS followed by COMMS\_ROUTING\_RECORDS. See the *LANSA for i User Guide* for detailed information.

LANSA Web running on the IBM i must be configured to allow interactive debug. This setting enables you to ensure that debug can only operate on your development or test system. The *LANSA Web Administrator* enables this setting to be enabled:

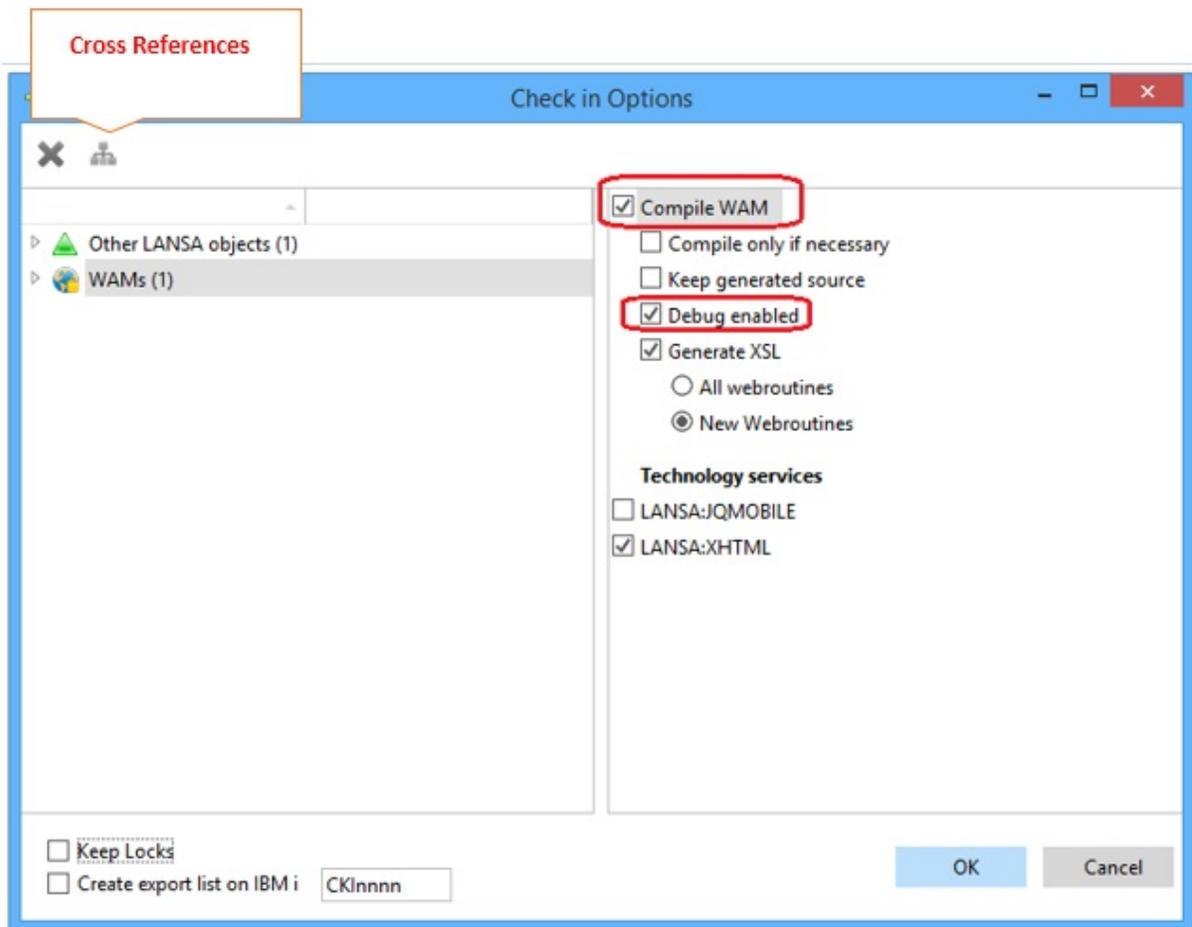


Restart LANSA Web after changing this setting. The Web Administrator Clean Up option will restart LANSA for the Web.



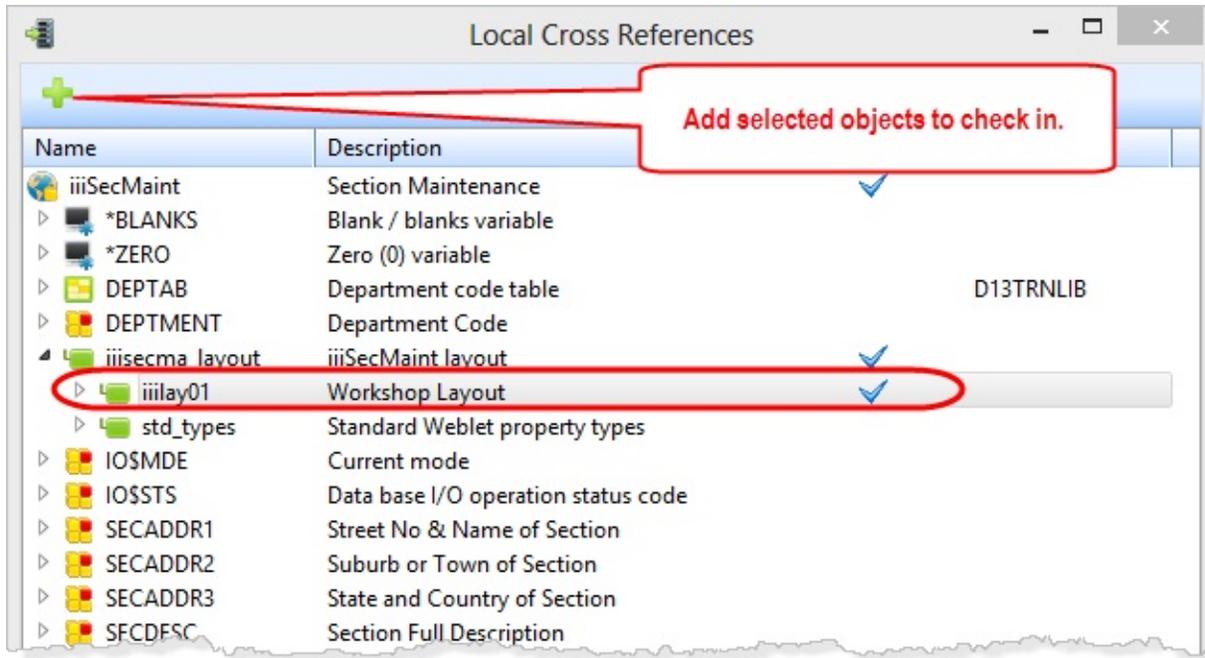
1. Check in your WAM **iiiSecMaint**. Its WAM layout, **iiisecma\_layout** and the

common WAM layout **iiilay01** will be included automatically. Check this by selecting the WAM and using the *Cross Reference* dialog.



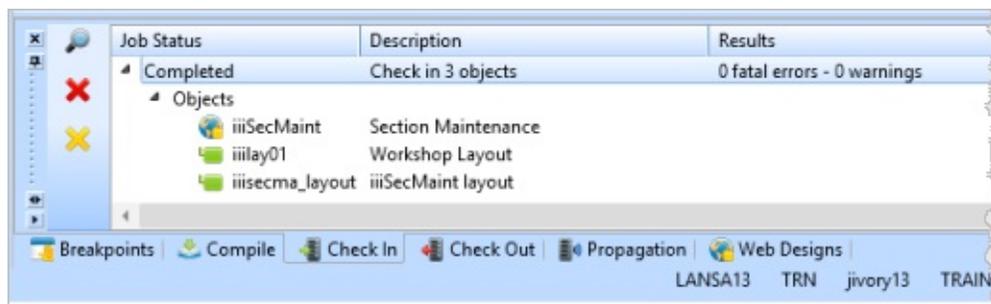
Make sure you check the options to compile the WAM and make it debug enabled.

2. In the *Local Cross References* dialog expand the WAM layout. `iiisecma_layout`.
  - a. Select the common layout, `iiilay01` and click the green cross button as shown, to add it to the list of objects to be checked in.

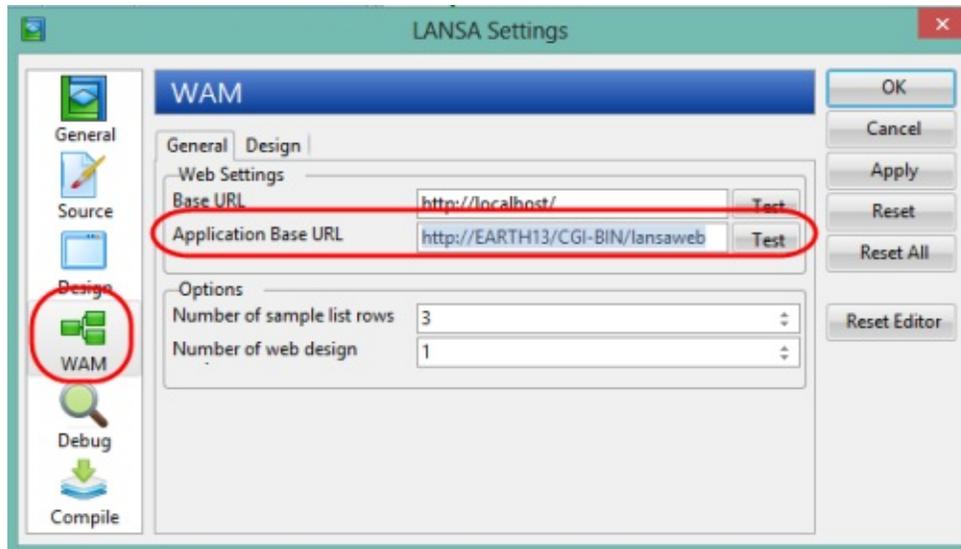


If required, you can select any other locally defined object and click the green cross button as shown to include these in the check in.

- Review the Check-in tab and ensure that your check-in and compile was successful.



- You now need to change your WAM execution settings, so that when you run the WAM from the Design view, it is run on the IBM i server.
  - From the File tab, open the *Editor Options* dialog and select the WAM settings.
  - Change the Application Base URL setting to point to the server name for your LANSAP IBM i system. Alternatively specify an IP Address.



- c. Use the *Test* button to check your entry is correct.
- d. Click OK to save your changes.

When you run your WAM it will now execute on the IBM i server.

5. From the *Design* view, Run your Begin WebRoutine to prove you can run the WAM on the IBM i server.
6. You should currently have the ADD\_ENTRY statement set as a break point. Run your WAM again, this time in Debug mode. You will now be able to debug your WAM exactly as before, when you were running the WAM locally.

## Summary

### Important Observations

- Your Visual LANSAs system, PC and LANSAs for the Web must be correctly configured to support interactive debug of WAMs. Refer to the [Interactive Debugging](#) in the *Web Administration Guide* for set up details.

### Tips and Techniques

- For debug support, make sure that the local Windows user, that is the default user for web jobs, is a member of the Group LANSAs and Administrator
- In the LANSAs Web Administrator for the local system, use the *Configure Data/Application Server* to make sure that the *Allow Interactive Debug* option is checked.
- For remote debug you need to use *LANSAs Web Administrator* for the IBM i server to *Allow Interactive Debug*. If you have a local DNS then no other changes are needed. Without a local DNS you need to configure the *LANSAs Listener* on the IBM I, adding an entry for each developer PC name and IP Address.
- Debug is supported via a Windows service running on the developer PC, which is started with Visual LANSAs.

### What You Should Know

- WAMs can be debugged interactively when run locally and also remotely when the WAM is run on the server (IBM i or Windows).

# WAM060 - Employee Maintenance using Advanced Weblets

## Objectives

To introduce a number of new weblets and techniques.

LANSA  
Advanced Software Made Simple

Department: ADMINISTRATOR DEPTXX Section: INTERNAL ADMIN SRV Search

Employ Number	Full Name	Post/zip Code	Home phone Number	Business Phone Number
A1001	JONES, BEN		2001 799 5268	798 0543
A1012	PAUL, PATRICK		2147 687 1717	222 2222
A1013	PATTISON, GEORGE		2010 222 2222	212 3569
A1015	WOODS, BRADLEY		2030 430 1236	489 2485
A1020	DOUGLAS, ADAM PETER		2147 674 5310	639 5188
A1021	MCCULLY, DAVID		2153 762 1321	159 6845
A1025	ROBINSON, MARY		2005 126 3598	456 1852
A1027	MORRISON, ALAN		2007 148 2365	489 2485
A1111	VEREY, WARREN PETER		2345 958 4567	957 3188
A1404	MRS BRICK, GILL		2090 3343 333	324 444
A1509	REDFORD, ROBERT		2060 5559966	9573188
A2000	BROWN, JOE		2100 01 888	01 888
A2002	BROWN, GEORGE		2000 020 1212	020 8899

LANSA  
Advanced Software Made Simple

Department: ADMINISTRATOR DEPTXX Section: INTERNAL ADMIN SRV Search

Employ Number	Full Name	Post/zip Code	Home phone Number	Business Phone Number
A1001	JONES, BEN		2001 799 5268	798 0543
A1012	PAUL, PATRICK		2147 687 1717	222 2222
A1013	PATTISON, GEORGE		2010 222 2222	212 3569
A1015	WOODS, BRADLEY		2030 430 1236	489 2485
A1020	DOUGLAS, ADAM PETER		2147 674 5310	639 5188
A1021	MCCULLY, DAVID		2153 762 1321	159 6845
A1025	ROBINSON, MARY		2005 126 3598	456 1852
A1027	MORRISON, ALAN		2007 148 2365	489 2485
A1111	VEREY, WARREN PETER		2345 958 4567	957 3188
A1404	MRS BRICK, GILL		2090 3343 333	324 444
A1509	REDFORD, ROBERT		2060 5559966	9573188
A2000	BROWN, JOE		2100 01 888	01 888
A2002	BROWN, GEORGE		2000 020 1212	020 8899

Details Skills

Employee Number A1001  
Employee Surname JONES  
Employee Given Name(s) BEN  
Street No and Name 144 Frog  
Suburb or dept PYMBLE.  
State and Country NSW.  
Post / Zip Code 2001  
Home Phone Number 799 5268  
Business Phone Number 798 0543  
Save

- Dynamic Select Boxes for department and sections are linked. The sections combo box is repopulated when department changes.
- The Search button populates a list of employees on the left hand side.
- The page is divided into two resizable areas by a Vertical Splitter weblet.
- A Details WebRoutine is called by selecting the hyperlink on employee number (an Anchor weblet)
- The Details WebRoutine outputs to the area on the right hand side defined by a Nav Panel.

- A Tab Pages weblet enables employee details and a list of skills to be shown on the Navigation panel.
- A Save button on each tab page enables employee details or skills to be updated.
- A Dynamic select box weblet is also used in skill code column of the employee skills list. A *New Skill* button adds a blank skill entry at the top of the list and the *Save* button then inserts a new skill.

To achieve these objectives you will complete the following:

Step 1. Create WAM iiiEmpMaint – Employee Maintenance

Step 2. Set up the ShowPage web page design

Step 3. Complete the ShowPage web page design

Step 4. Define the Details WebRoutine

Step 5. Extend the Details WebRoutine for update

Step 6. Extend the Details WebRoutine to add new employee skill

Step 7. Control which Tab is redisplayed

Step 8. Replace Date Acquired with a Date field (Optional).

Step 9. Change Grade to a Dropdown list (Optional)

Summary

## **Before You Begin**

Complete all preceding exercises in this workshop.

## Step 1. Create WAM iiiEmpMaint – Employee Maintenance

### 1. Create a new WAM:

Name: iiiEmpMaint

Description: Employee Maintenance

Layout Weblet: iiilay01

### 2. Begin by defining the lists needed to support the main page WebRoutine, **ShowPage**.

- Define a working list, DEPTS for fields in the file DEPTAB
- Define a working list, SECTS for section code and description from file SECTAB

These lists will support Dynamic Select Box weblets for department and section codes.

- Define a working list, EMPLOYS for fields EMPNO, FULLNAME, POSTCODE, PHONEBUS, and PHONEHME. All fields should be defined with an output attribute. This list of employees will be displayed on the left hand side of the vertical splitter weblet.
- Define a Group\_by, EMPS for fields EMPNO, SURNAME, GIVENAME, POSTCODE, PHONEBUS, PHONEHME
- Map the field STDRENTY for both, as a hidden field

Your code should look like the following:

```
Function Options(*direct)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
* Support web page ShowPage
Def_List Name(#depts) Fields(#deptment #deptdesc) Type(*Working)
Def_List Name(#sects) Fields(#section #secdesc) Type(*Working)
Def_List Name(#employs) Fields((#empno *out) (#fullname *out) (#postcode
Group_by Name(#emps) Fields(#empno #surname #givename #postcode #pho
Web_Map For(*both) Fields((#stdrenty *hidden))
End_Com
```

### 3. Define a ShowPage WebRoutine.

- Define a web\_map for output for lists DEPTS, SECTS, mapped as JSON data and list EMPLOYS.

- Define a web\_map for both, for fields DEPARTMENT and SECTION
- Define a CASE loop for field STDREENTRY
- When = S

Clear list EMPLOYEES

Select fields in EMPLOYEES from file PSLMST1 with key DEPARTMENT and SECTION

Fullname = Surname + Givenname

Add entry to EMPLOYEES

End select

- End Case

Your code should look like the following:

```
WebRoutine Name(ShowPage)
Web_Map For(*output) Fields((#depts *json) (#sects *json) #employs)
Web_Map For(*both) Fields(#department #section)
Case (#stdreentry)
When (= S)
Clr_list #EMPLOYEES
Select Fields(#emps) From_File(pslmst1) With_Key(#department #section)
#fullname := #surname + ', ' + #givenname
Add_Entry To_List(#employs)
Endselect
Endcase
Endroutine
```

4. Define a method routine **BuildDepts** to populate list DEPTS.

The routine should:

Define an input parameter, named i\_dept, based on DEPARTMENT

Clear the list

Select all entries from file DEPTAB, and add to list

If i\_dept is blank, position to the first entry

Else set DEPARTMENT to I\_DEPT

5. Define a method routine **BuildSects** to populate list SECTS.

The routine should:

Define an input parameter, named i\_dept, based on DEPARTMENT

Clear the list

Select entries from file SECTAB with a key of I\_DEPT and add to list

Position to the first entry

Your new code should look like the following:

```
Mthroutine Name(BuildDepts)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Clr_List Named(#depts)
Select Fields(#depts) From_File(deptab)
Add_Entry To_List(#depts)
Endselect
If (#i_dept = *blanks)
Get_Entry Number(1) From_List(#depts)
Else
#deptment := #i_dept
Endif
Endroutine

Mthroutine Name(BuildSects)
Define_Map For(*input) Class(#deptment) Name(#i_dept)
Clr_List Named(#septs)
Select Fields(#septs) From_File(sectab) With_Key(#i_dept)
Add_Entry To_List(#septs)
Endselect
Get_Entry Number(1) From_List(#septs)
Endroutine
```

6. Invoke these method routines at the end of the WebRoutine **ShowPage**

Your code should look like the following. New code is shown in red.

```
.....
#com_owner.BuildDepts I_Dept(#deptment)
#com_owner.buildsepts I_Dept(#deptment)
Endroutine
```

7. The Dynamic Select Box weblet will be set up to invoke a response WebRoutine to re-populate the list SECTS when field DEPARTMENT changes.

This will require a WebRoutine, UpdSects defined as follows:

The WebRoutine must be defined with a Response() keyword with the value SON

Map for input field DEPARTMENT

Map for output the list SECTS as \*JSON data

Invoke the BuildSects method routine, passing DEPARTMENT

Your code should look like the following:

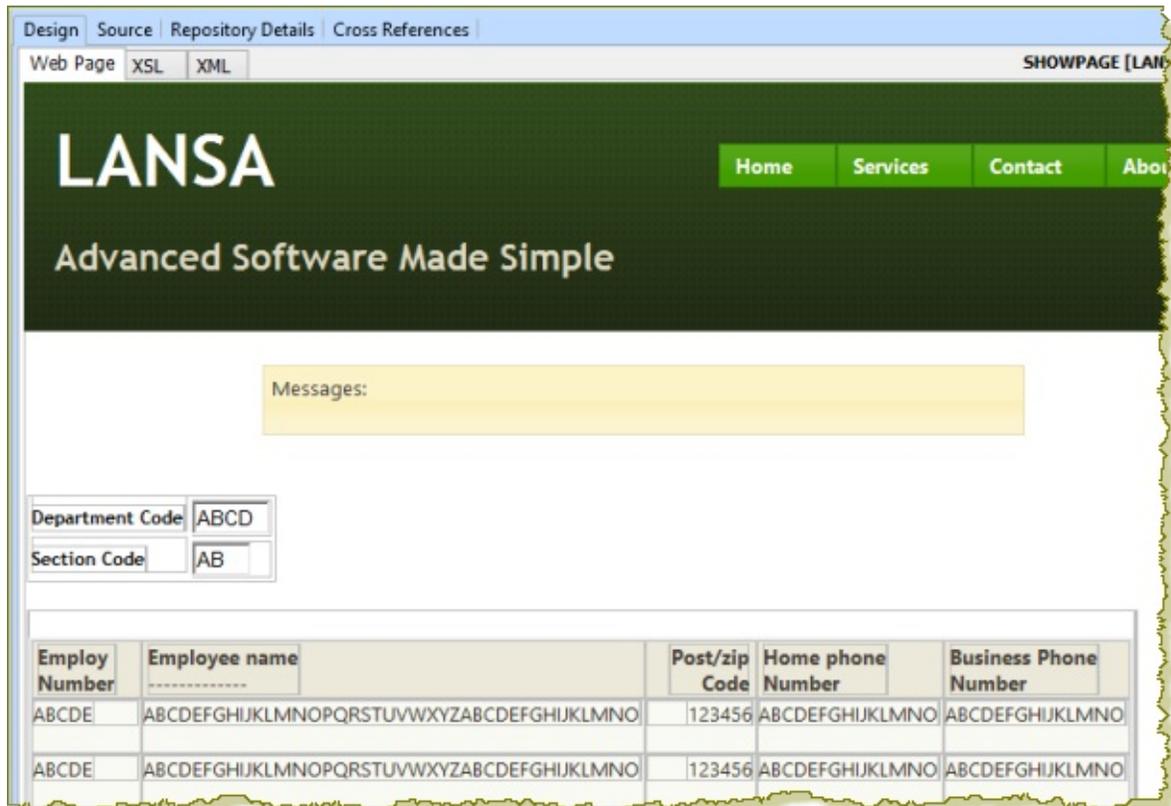
```
WebRoutine Name(updsects) Response(*JSON)
Web_Map For(*input) Fields(#deptment)
Web_Map For(*output) Fields((#sects *json))
#com_owner.BuildSects I_Dept(#deptment)
Endroutine
```

8. Compile your WAM.

## Step 2. Set up the ShowPage web page design

1. Open the **ShowPage** WebRoutine in the *Design* view.

It should look like the following:



2. Click anywhere in the table containing department code and using the context menu select the *Table Items / add Columns...* option, to add 3 columns.
  - a. Move the section code and its label into the 3<sup>rd</sup> and 4<sup>th</sup> columns (top row), using drag and drop.
  - b. Drop a push button with image into the 5<sup>th</sup> cell, top row
  - c. Click in the bottom row, and use the context menu, *Table Items / Delete Row* to delete this row.
  - d. Click in the cell containing the *section code* label, if necessary use the cursor keys to ensure you are in the table cell (<td> tag) and use the *Details* tab to change its *class* to *caption*.
  - e. Select the *department code* label and delete it. Type Department: into the

cell. Change the cell *class* to **caption**.

f. Select the *section code* label and delete it. Type Section: into this cell.

g. Select each of the new cells using the cursor keys, and delete the \* place holder characters.

h. Save your changes.

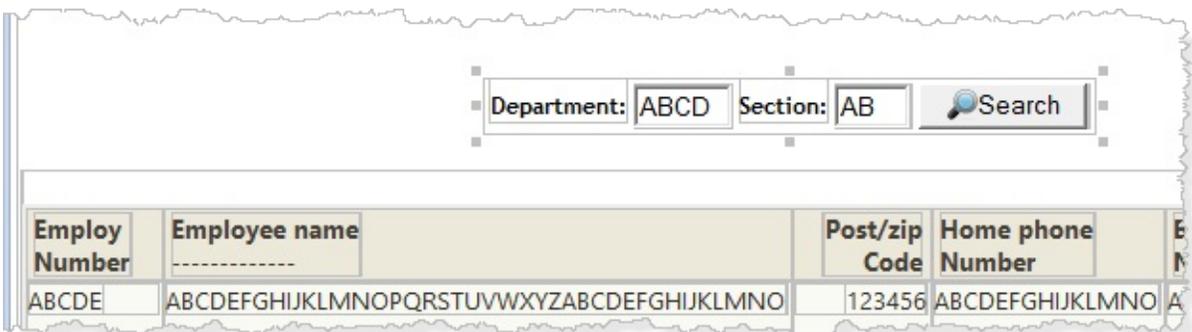
3. Select the push button and set up its properties on the Details tab:

Property	Value
caption	<b>Search</b>
left_relative_image	<b>icons/normal/16/zoom_16.png</b>
on_click_wname	<b>ShowPage</b>
submitExtraFields	<b>Field Name: STDRETRY</b>
	<b>Literal Value: S</b>

4. Select the table and changes its *Align* property to center. You should be able to click on a corner of the table to select it.

5. Save your changes

Your page should look like the following:



6. Drag and drop a Dynamic Select Box onto the field DEPARTMENT, set up its properties as shown:

Property	Value
listname	<b>DEPTS</b>

codeField	<b>DEPARTMENT</b>
captionField	<b>DEPTDESC</b>

Adjust the width of the dynamic select box.

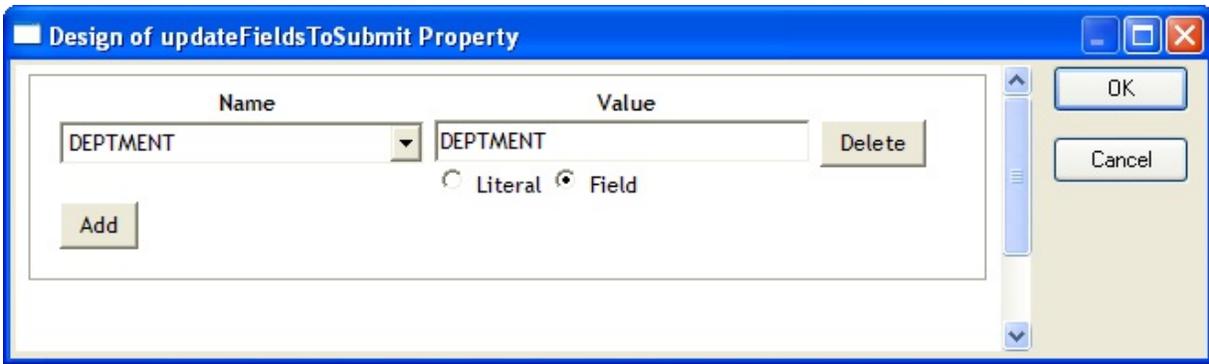
7. Drag and drop a Dynamic Select Box onto the field SECTION and set up its properties as shown:

<b>Property</b>	<b>Value</b>
Listname	<b>SECTS</b>
codeField	<b>SECTION</b>
captionField	<b>SECDESC</b>
updateWrName	<b>updsects</b>
updateOnFieldChange	<b>DEPARTMENT</b>
updateFieldsToSubmit	<b>Field Name: DEPARTMENT</b>
	<b>Value: Field DEPARTMENT</b>

Adjust the width of the dynamic select box.

Complete the *updateFieldsToSubmit* property by clicking on the Ellipsis  button in the value column, to open the *Design of...Property* dialog.

- a. Select DEPARTMENT in the *Name* column.
- b. Select the *Field* checkbox and enter **DEPARTMENT** in the *Value* column.



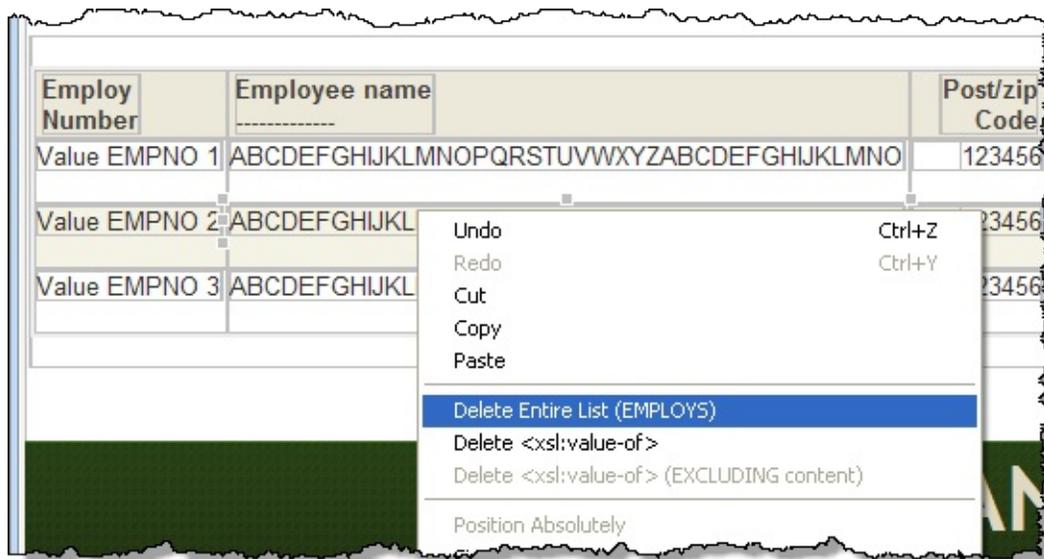
8. Save your changes.
9. Execute the **ShowPage** WebRoutine in the browser.
  - Changing selected department should refresh the section's dropdown list.
  - The Search button should populate the list of employees.



### Step 3. Complete the ShowPage web page design

In this step you will:

- Add a Vertical Splitter weblet to the page and move the employees list onto its left hand side.
  - Add a Nav Panel to the right hand side of the vertical splitter and give this a name
  - Add an Anchor weblet to the employee number column in the employees list and set the weblet up to invoke a **Details** WebRoutine and output to the nav panel.
  - You will create the **Details** WebRoutine in the next step.
1. In this step you will temporarily remove the employees list from the page. With the **ShowPage** WebRoutine open in the Design view, select anywhere inside the employees list and use the context menu, to select the *Delete Entire List (EMPLOYS)* option:



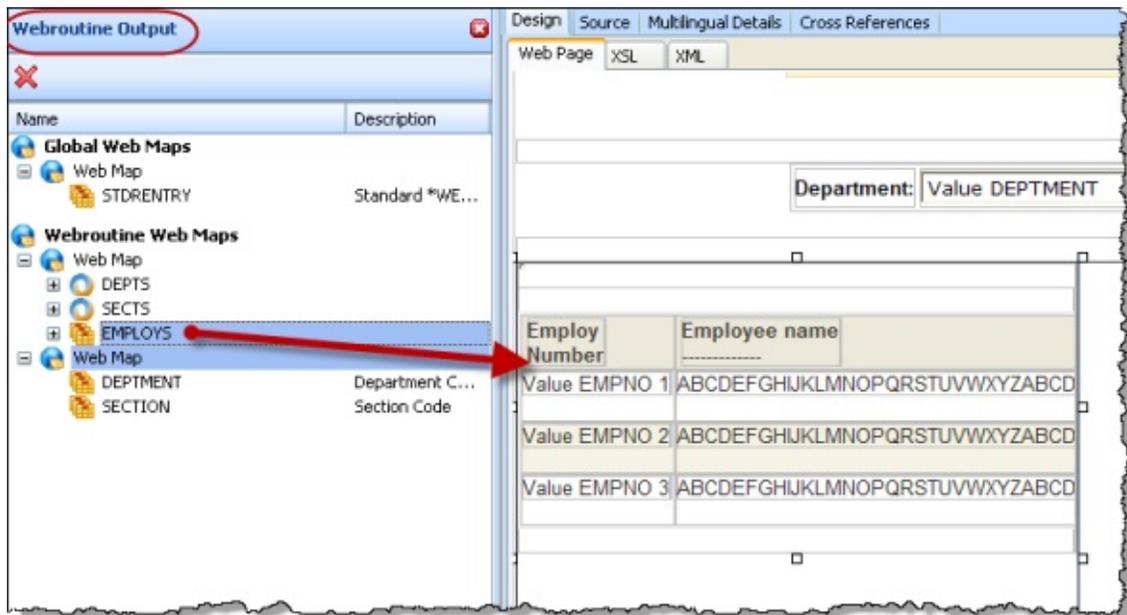
2. Drag and drop a Vertical Splitter onto the page.

Set up its properties as:

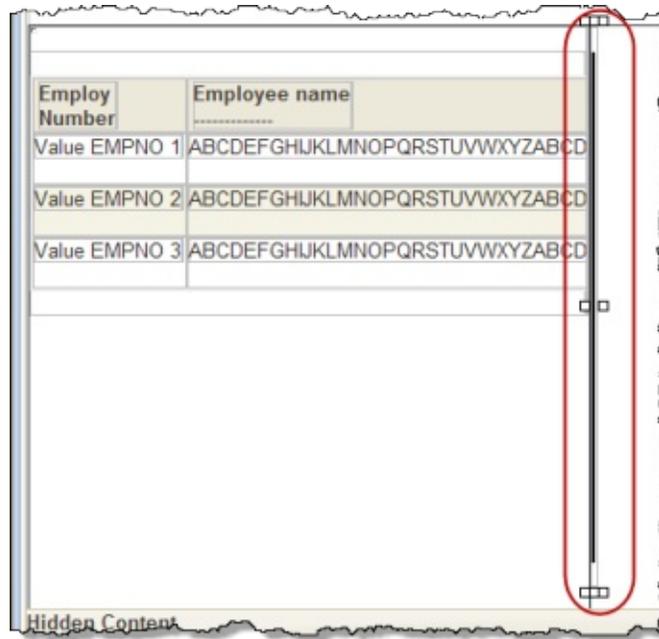
Property	Value
Width	100%
Height	500px
	40

Left\_proportion\_percent

3. Select the *WebRoutine Output* tab, and drag and drop the list EMPLOYS onto the left side of the Vertical Splitter:



4. Save your changes.
5. Drag and drop a Navigation Panel onto the right side of the vertical splitter.



With the Navigation Panel selected set up its properties:

Property	Value
name	<b>EmpDtl</b>
border	<b>none</b>
border_width	<b>0px</b>
height	<b>450px</b>

6. Drag and drop an Anchor weblet into the Employ Number column in the employees list. With the Anchor weblet selected, set up its properties as:

Property	Value
Currentrowhfield	<b>EMPNO</b>
Currentrownumvalue	<b>\$EMPNO</b>
Rentryvalue	<b>D</b>
On_click_wname	<b>Details</b>
target_window_name	<b>EmpDtl</b>

Type in the Details WebRoutine name, because you haven't yet defined it.

7. Save your changes.

## Step 4. Define the Details WebRoutine

The **Details** WebRoutine:

- Will be invoked by selecting an employee in the employees list, using the Anchor weblet (also known as a hyperlink). Output for the **Details** WebRoutine will be shown on the **ShowPage** web page.
  - Output will be displayed in the Navigation Panel on the right of the Vertical Splitter.
  - Will handle display and update of employee and employee skills data. It will also handle adding a new skill for the employee.
1. At the top of your WAM, begin by defining fields, lists or group\_by that will be required to support the **Details** WebRoutine.
    - a. Define a working list, EMPSKLS for employee skills containing SKILCODE, GRADE, COMMENT, DATEACQ, DATEACQR and EMPNO . DATEACQR and EMPNO should be hidden.
    - b. Define a work field EMPNOW based on EMPNO
    - c. Define a group\_by, EMPDATA containing fields EMPNOW, SURNAME, GIVENAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, PHONEBUS and PHONEHME. EMPNOW should be output.

Your code should look like the following:

```
Def_List Name(#empskls) Fields((#SKILCODE *out) #GRADE #COMMENT  
*  
Define Field(#empnow) Reffld(#empno)  
Group_By Name(#empdata) Fields((#empnow *out) #SURNAME #GIVENAI
```

2. Create a **Details** WebRoutine. Initially you will create a simple version of this WebRoutine and then extend it.
  - a. Map field EMPNO for both as a hidden field.
  - b. Map for both, Group\_by, EMPDATA and list EMPSKLS. EMPSKLS should be mapped with a \*private attribute. \*Private means it will not be automatically shown on the web page.

Your code should look like the following:

```
WebRoutine Name(Details)  
Web_Map For(*both) Fields((#empno *hidden))
```

```
Web_Map For(*both) Fields(#empdata (#empskls *private))  
Endroutine
```

3. In this step you will add logic to your **Details** WebRoutine using a CASE loop for STDREENTRY (this is mapped globally as a hidden field).

**Note the following:**

Field EMPNOW is shown on the Details web page as an output field. EMPNO is mapped as a hidden field and will be used for employee update. EMPNO is passed to the Details WebRoutine by the Anchor weblet.

Code your **Details** WebRoutine based on the following:

- Change EMPNOW to EMPNO
- Case loop on STDREENTRY
- When = D

Fetch fields in EMPDATA from file PSLMST with key EMPNO

Clear list EMPSKLS

Select EMPSKLS from file PSLSKL with key EMPNO

Add entry to EMPSKLS

Endselect

- When = U

Update fields in EMPDATA in file PSLSMT with key EMPNO. Go to next statement on validation error.

If status is not OK, issue an error message

Else

Issue an "employee changed" message including employee number.

Endcase

Your code should look like the following:

```
WebRoutine Name(Details)  
Web_Map For(*both) Fields((#empno *hidden))  
Web_Map For(*both) Fields(#empdata (#empskls *private))  
Web_Map For(*both) Fields((#skilcode *hidden))  
#empnow := #empno  
Case (#stdreentry)
```

```

When (= D)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Clr_List Named(#empskls)
Select Fields(#empskls) From_File(pslskl) With_Key(#empno)
Add_Entry To_List(#empskls)
Endselect
* update employee
When (= U)
Update Fields(#empdata) In_File(pslmst) With_Key(#empno) Val_Error(*next
If_Status Is_Not(*okay)
Message Msgtxt('Error occurred on employee update')
Else
Message Msgtxt(' Employee ' + #empno + ' was changed')
Endif
Endcase
Endroutine

```

4. Compile your WAM and open the **Details** WebRoutine in the *Design* view.

Your web page should look like the following:

Employee Number	ABCDE
Employee Surname	ABCDEFGHIJKLMNOPQRS
Employee Given Name(s)	ABCDEFGHIJKLMNOPQRS
Street No and Name	aAbBcCdDeEfFgGhHiIjJkKlLm
Suburb or Town	aAbBcCdDeEfFgGhHiIjJkKlLm
State and Country	aAbBcCdDeEfFgGhHiIjJkKlLm
Post / Zip Code	123456
Home Phone Number	ABCDEFGHIJKLMNO
Business Phone Number	ABCDEFGHIJKLMNO

5. Drag and drop a *Tab Pages* weblet (not *Tab Pages (deprecated)*) onto the

page.

A screenshot of a web form with the following fields and values:

Employee Number	ABCDE
Employee Surname	aAbBcCdDeEfFgGhHij.
Employee Given Name(s)	aAbBcCdDeEfFgGhHij.
Street No and Name	aAbBcCdDeEfFgGhHijJkKIL
Suburb or Town	aAbBcCdDeEfFgGhHijJkKIL
State and Country	aAbBcCdDeEfFgGhHijJkKIL
Post / Zip Code	123456
Home Phone Number	ABCDEFGHijklmI
Business Phone Number	ABCDEFGHijklmI

At the bottom, there are three tabs: Tab Page 1 (selected), Tab Page 2, and Tab Page 3.

- a. Click on the Tab Pages weblet to select it. On the *Details* tab set up its *tabs* properties by clicking on the *Ellipsis*  button, that will open the *Design of... Properties* dialog:

The 'Design of tabs Property' dialog box shows a list of tabs: Tab Page 1, Tab Page 2, and Tab Page 3. To the right of the list are buttons for 'Move Up', 'Move Down', 'Add New', and 'Remove (DEL)'. Below the list is the 'Tab Properties' section with the following fields:

- Caption: Tab Page 1 (selected)
- Or \*MTXT Variable: (empty)
- Image: (empty)
- Selected image: (empty)
- Hide if true: (dropdown)
- Disable if true: (dropdown)
- Preload Nav Panels
- Reload Nav Panels on tab click

Tip: Select an item in the list to edit it.

- b. Select each tab page entry in the list and set up its properties:

Property	Value
caption	Details

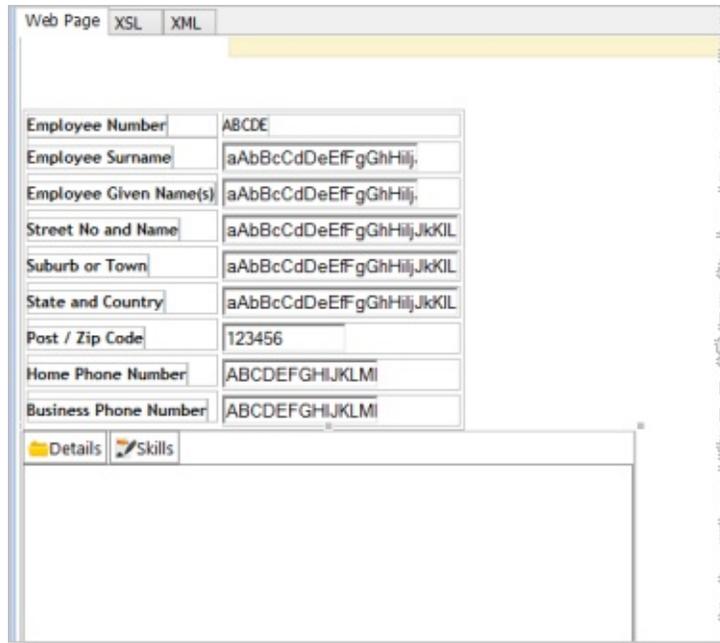
image	icons/normal/16/folder_16.png
-------	-------------------------------

Property	Value
caption	Skills
image	icons/normal/16/contract_16.png

- c. Delete Tab Page 3.
  - d. Save your changes by clicking OK.
6. Click on the *Tab Pages* weblet to select it and set its height and width properties:

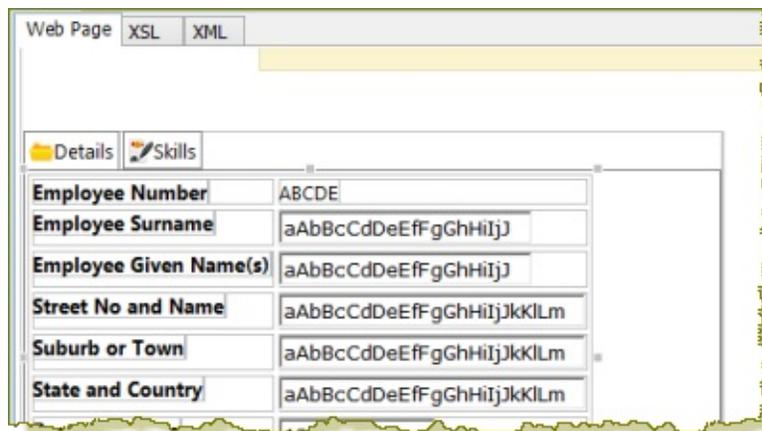
Property	Value
content_width	450px
content_height	400px

7. Save your changes. Your page should now look like the following:



8. Click on a corner of the employees fields table to select it. Use the context menu to select *Cut*.
  - a. Click on the Tab Pages weblet to select it.
  - b. Click on the *Details* tab to ensure the correct tab sheet is selected.
  - c. Use the context menu to *Paste* the table into the tab folder *Details* tab.

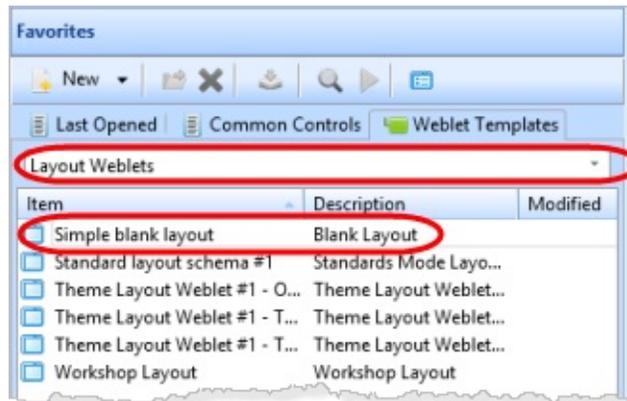
Your page should now look like the following:



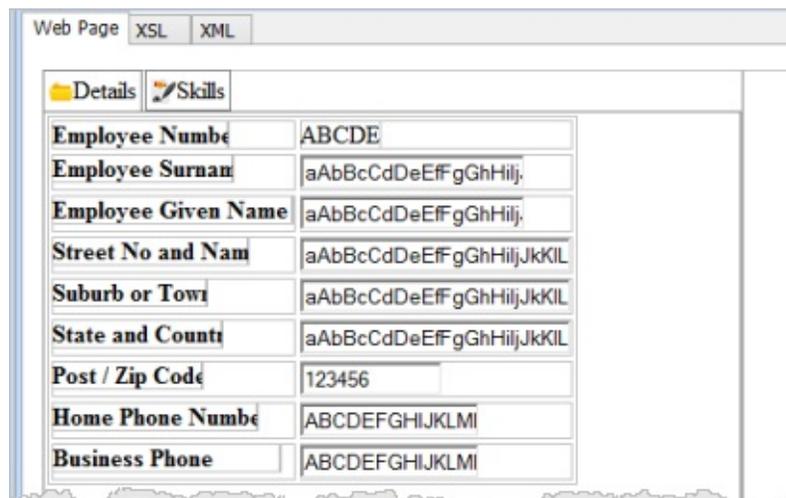
9. Save your changes.
10. Click on the Skills tab to select it and drag and drop a *jQuery Enabled Grid* onto the *Skills* tab page.
11. Select the Grid and set up its properties:

Property	Value
Listname	EMPSKLS

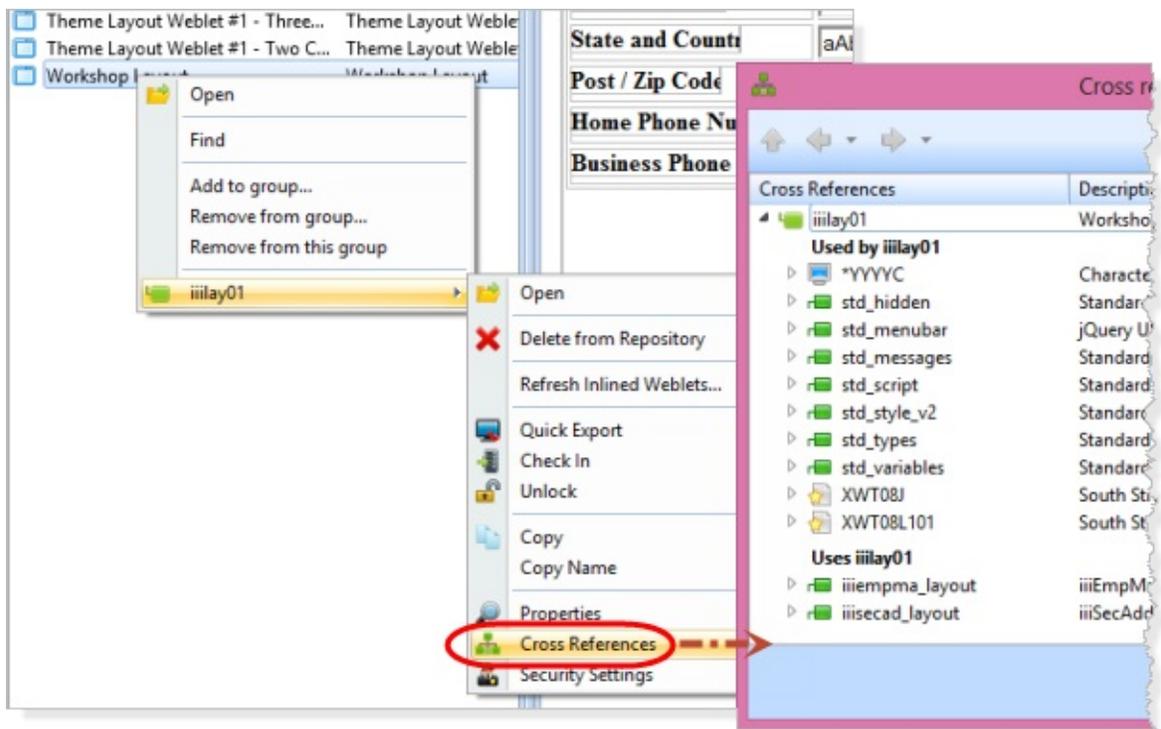
12. Click on the *Details* tab page and save your changes. This will make *Details*, the default tab at run time.
13. The **Details** web page will be displayed inside the *Navigation Panel* on the **ShowPage** web page. The **Details** web page therefore requires a blank layout.
  - a. On the *Favorites / Weblet Templates* tab, using the dropdown at the top, select *Layout Weblets*:



- b. Drag and drop the *Simple blank layout* onto the page. Your design should now look like the following:



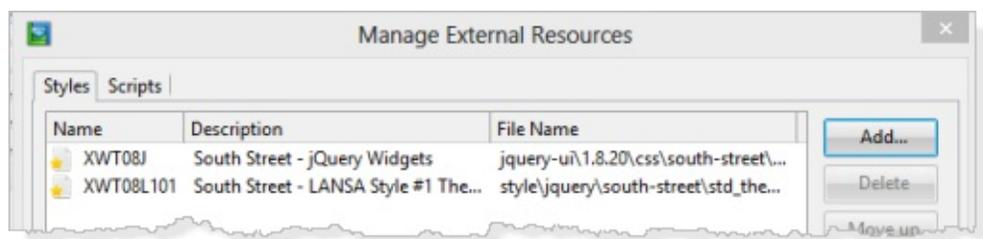
14. You need to ensure that this blank page adopts the same theme as your common layout **iiilay01**.
- On the *Favorites / Weblet Templates* tab, select *Layout Weblets* in the top dropdown list.
  - Right click on the layout *Workshop Layout* and use the context menu to select *Cross References*:



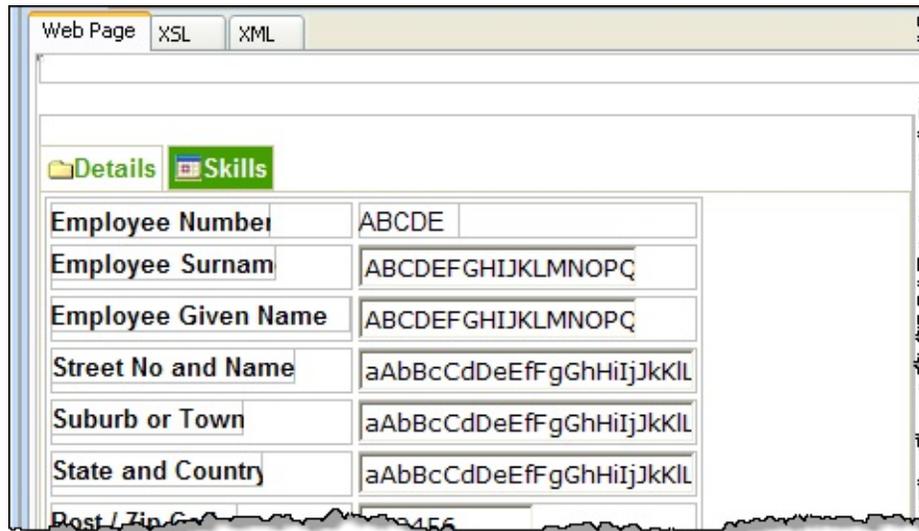
- Make a note of the *Style External Resources* being used. In this example these are XWT08J and XWT08L1.

Close this dialog.

- In the *Design* view for WebRoutine **Details**, on the Design ribbon, select the *External Resources* button. Use the *Add* button to add the two Styles required.



e) Your *Design* should now look like the following, but reflecting your chosen theme:



Note the Font used for the field labels.

Note also the background used for a non-selected tab. The color may not be shown in the *Design* view. Check your results in the next step.

15. Save your changes.

16. Execute the **ShowPage** WebRoutine in the browser to test your application:

- Selecting an employee should display details on the right side of the Vertical Splitter. The *Details* tab will be shown initially, as it was saved as the default.
- Selecting the *Skills* tab will display a grid of employee skills.
- Selecting another employee will display the *Details* tab again.

Later you will enhance the application to remember the current tab and display it for the next employee selected.

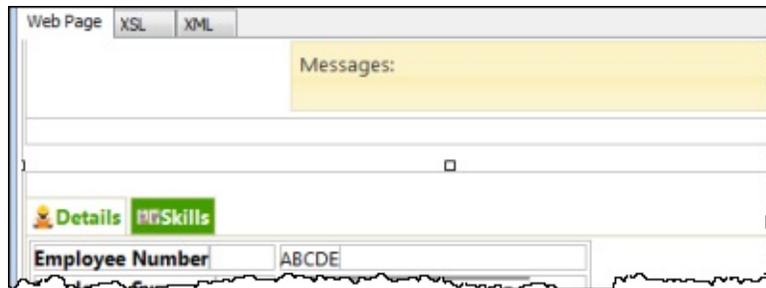
## Step 5. Extend the Details WebRoutine for update

In this step you will:

- Make the **Details** tab handle update.
  - Extend the **Details** WebRoutine to handle update of Employee skills
  - Add a *Save* button to the *Skills* tab.
1. Open the **Details** WebRoutine in the *Design View*. The *Details* tab should be shown. If necessary select the *Details* tab and save your WAM to make this the default.
  2. On the *Details* tab, select anywhere in the table containing the fields and use the context menu to select *Table Items/Add Rows...* to add 1 row to the bottom of the table.
  3. Click in the left hand cell of the new row and set its *align* property to left.
  4. Drag and drop a *Push Button with Image* into the bottom left cell of the table. Set up the button as follows:

<b>Property</b>	<b>Value</b>
Caption	<b>Save</b>
left_relative_image	<b>icons/normal/16/check_mark_16.png</b>
on_click_wrname	<b>Details</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal value: U</b>

5. This step adds a *Message* weblet to this page:
  - a. Select the tab pages weblet. Move the cursor right and press enter. This should create a blank space above the tab pages weblet.
  - b. Drag and drop a *Messages* weblet into the space immediately above the *Tab Pages* weblet. Your design should now look like the following:



6. Select the Messages weblet and set its *target\_window\_name* property to **\_top**. This will route messages from this page to the messages weblet on the page in which it is embedded.

Save your changes.

7. Retest your WAM. The update employee details logic has already been included in the **Details** WebRoutine. You should now be able to change employee data and save the changes. Try making an invalid change, such as a blank surname.

8. To handle update of employee skills the CASE loop in WebRoutine Details requires new logic.

The employee skills list EMPSKLS is mapped for both. The list should be read process updates to the employee skills file PLSKLS.

The SKILCODE in EMPSKLS is mapped for \*output. In order to update the employee skill record, this key value will need to be saved in a hidden field in the t.

When the list is read (SELECTLIST) the key value SKILCODE will be set from a hidden field.

The list contains a hidden field - DATEACQR. This will be used to recognize list tries for existing records. This will be important when you extend the logic to handle insert of a new employee skill.

- a. Define a work field SC based on field SKILCODE
- b. Add field SC to list EMPSKLS as a hidden field. Your code should look like the following:

**Define Field(#sc) Reffld(#skilcode)**

```
Def_List Name(#empskls) Fields((#SKILCODE *out) #GRADE
#COMMENT #DATEACQ (#dateacqr *hidden) (#empno *hidden) (#sc
*hidden)) Type(*Working) Entrys(*max)
```

- c. Change the existing logic which builds the employee skills list, to

populate the hidden field SC. Your code should look like the following.  
The new code is shown in red.

```
Select Fields(#empskls) From_File(pslskl) With_Key(#empno)
#sc := #skilcode
Add_Entry To_List(#empskls)
Endselect
```

d. Add new logic to update employee skills based on the following:

When = S

Read list EMPSKLS using SELECTLIST/ENDSELECT

Change SKILCODE to SC

If DATEACQR is not \*zeroes

Update fields in list EMPSKLS in file PLSKL with key EMPNO and SKILCODE. Go to next line on validation error.

End if

End select

If status is OK, issue message, employee 'Skills for nnnn were changed', end if, where nnnn is employee number.

Your code should look like the following:

```
When (= S)
```

```
Selectlist Named(#empskls)
```

```
#skilcode := #sc
```

```
If (#dateacqr *NE *zeroes)
```

```
Update Fields(#empskls) In_File(pslskl) With_Key(#empno #skilcode) Val_Er
```

```
Endif
```

```
Upd_Entry In_List(#empskls)
```

```
Endselect
```

```
If_Status Is(*okay)
```

```
Message Msgtxt('Skills for ' + #empno + ' were changed')
```

```
Endif
```

```
Endcase
```

8. Compile your WAM.

9. Open the **Details** WebRoutine in the *Design* view.

- a. On the Skills tab, select the grid and move to the right using the cursor keys and press enter. This will position the cursor immediately below the grid.
  - b. Use the context menu, *Insert HTML, Table* to insert a table with one row and two columns.
10. Click in each cell of the new table and change the align property to left.
11. Drag and drop a Push Button with Image into the right hand cell and set it up as follows:

<b>Property</b>	<b>Value</b>
caption	<b>Save</b>
left_relative_image	<b>icons/normal/16/check_mark_16.png</b>
on_click_wname	<b>Details</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: S</b>

12. Remove the \* place holder characters from the right hand cell. Leave them in the left cell.
13. Click on the *Details* tab page to select it and then Save your changes.
14. Re-test your WAM. Note that on the *Home* ribbon, you can use *History* button in the *Runtime* gallery, to re-run any VL component.



- You should be able to change any employee skills data and save changes with the *Save* button.
- Errors such as *Date Acquired = zero* or other validation errors will display messages in the messages weblet.
- After the *Save* button is processed the *Details* tab is redisplayed, because this is the default tab. You will fix this issue in a later enhancement.

## Step 6. Extend the Details WebRoutine to add new employee skill

In this step you will extend the WebRoutine Details to insert a new employee skill.

- A *New Skill* push button on the Skills tab will refresh the grid with a blank first entry.
  - The SKILCODE will now be an input field in the list EMPSKLS and the hidden field SC will no longer be needed.
  - The SKILCODE column in the grid will be customized using a Dynamic select box.
  - The Dynamic select box will be populated with a list of skills from the table SKLTAB. A new method routine will build this list.
  - When the Skills Save button is processed, an employee skill will be inserted if the DATEACQR field is zero
1. Change the definition of list EMPSKLS so that SKILCODE is input capable, removing the hidden field SC. Your code should look like the following:  

```
Def_List Name(#empskls) Fields(#SKILCODE #GRADE #COMMENT  
#DATEACQ (#dateacqr *hidden) (#empno *hidden)) Type(*Working)  
Entrys(*max)
```
  2. Delete the definition of field SC and remove all code which refers to it.
  3. Define a Group\_by SKL\_LIST for fields SKILCODE, GRADE, COMMENT, DATEACQ, DATAECQR. This will be used to clear employee skills list fields before adding a blank entry for insert.
  4. Define a working list SKILLS containing fields SKILCODE, SKILDESC. This list will mapped for output to populate the skill code Dynamic select box.

LANSA definition statements can be placed anywhere in your code. It is usual to place them at the top of the program code.

Your new code should look like the following:

```
Group_By Name(#skl_list) Fields(#skilcode  
#grade #comment #dateacq #dateacqr)  
Def_List Name(#skills) Fields(#skilcode #skildesc) Type(*Working)  
Entrys(*max)
```

5. Add a new When clause to the **Details** WebRoutine, that will handle a request from the *New Skill* button to add a blank entry as the first entry in the employee skills list. This will allow insert of a new employee skill.

The logic should be based on the following:

- When = N

Change SKL\_LIST to default values

Add entry to EMPSKLS after \*Start

Message 'Complete new skill in the first list entry'

Your new code should look like the following:

```
* add new top row to skills list
```

```
When (= N)
```

```
#skl_list := *default
```

```
Add_Entry To_List(#empskls) After(*START)
```

```
Message Msgtxt('Complete new skill in the first list entry')
```

6. When processing a list entry, a value of zero for Date Acquired (DATEACQR) means the entry is new.

Add the following new logic to the CASE loop, for when STDREENTRY = S.

New code is shown in red.

```
When (= S)
```

```
Selectlist Named(#empskls)
```

```
If (#dateacqr *NE *zeroes)
```

```
Update Fields(#empskls) In_File(pslskl) With_Key(#empno #skilcode) Val_Er
```

```
Else
```

```
Insert Fields(#empskls) To_File(pslskl) Val_Error(*next)
```

```
Endif
```

```
Endselect
```

```
If_Status Is(*okay)
```

```
Message Msgtxt('Skills for ' + #empno + ' were changed')
```

```
Endif
```

7. Create a method routine **BuildSkills** to populate the skills list SKILLS.
  - a. Clear the list SKILLS
  - b. Select all records from file SKLTAB and add entries to list SKILLS

c. Position to the first entry

Your code should look like the following:

```
Mthroutine Name(BuildSkills)
Clr_List Named(#skills)
Select Fields(#skills) From_File(skltab)
Add_Entry To_List(#skills)
Endselect
Get_Entry Number(1) From_List(#skills)
Endroutine
```

8. Add an output web\_map for the SKILLS list to the **Details** WebRoutine as JSON data. Your code should look like the following:

```
Web_Map For(*output) Fields((#skills *JSON))
```

9. Invoke the BuildSkills method at the end of the **Details** WebRoutine. Your code should look like the following. New code is shown in red.

```
Endcase
```

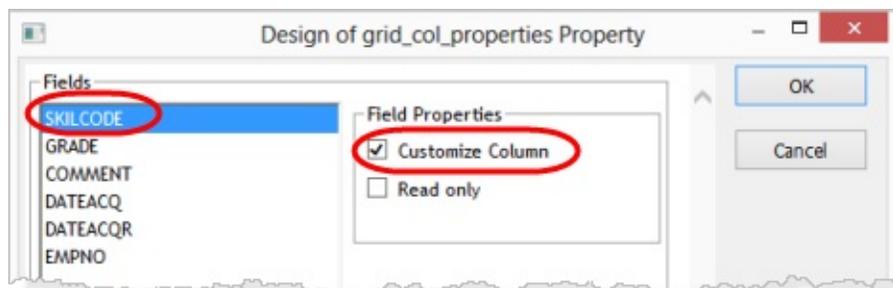
```
#com_owner.BuildSkills
```

```
Endroutine
```

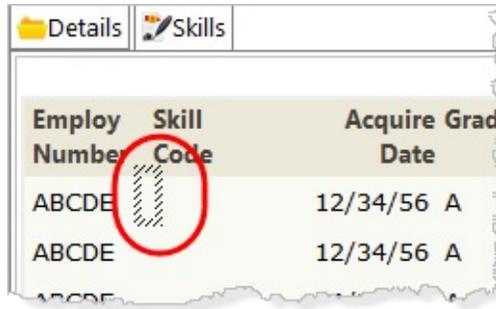
10. Compile your WAM.

11. Open the **Details** WebRoutine in the *Design* view.

12. Select the Skills tab, and select the Grid weblet. On the *Details* tab use the Ellipsis button for the *grid\_col\_properties* value to open the *Design of ....* dialog. With SKILCODE field selected, select the *Customize Column* check box and then click *OK* to close the dialog.



13. Drop a Dynamic select box into the Skill Code column. See the image below showing how the editor will highlight the column when the cursor is in the correct position.



Adjust the width of the dropdown so that it can display skill description.

14. Select the Dynamic select box and set up its properties as shown:

Property	Value
listName	<b>SKILLS</b>
codeField	<b>SKILCODE</b>
captionField	<b>SKILDESC</b>

15. Add a push button with image weblet into the into the single row table below the grid. Set up the button properties as:

Property	Value
caption	New Skill
left_relative_image	icons/normal/16/contract_16.png
on_click_wname	Details
submitExtraFields	Field Name: STDREENTRY
	Literal Value: N

Adjust the width of the push button to display the caption as single line.

Remove the place holder characters from table cell.

16. Select the Details tab page and then Save your changes. Your Skills tab page

should look like the following:

The screenshot shows a software interface with two tabs: 'Details' and 'Skills'. The 'Skills' tab is active, displaying a table with the following columns: 'Employ Number', 'Skill Code', 'Acquire Date', 'Grade', and 'Comment'. The table contains three rows of data. Below the table, there are two buttons: 'Save' (with a green checkmark icon) and 'New Skill' (with a pencil icon). The table data is as follows:

Employ Number	Skill Code	Acquire Date	Grade	Comment
ABCDE	ABCDEFGHIJ	12/34/56	A	aAbBcCdDe
ABCDE	ABCDEFGHIJ	12/34/56	A	aAbBcCdDe
ABCDE	ABCDEFGHIJ	12/34/56	A	aAbBcCdDe

17. Retest your WAM. Select an employee and select the Skills tab. The Dynamic select box for skills should display the correct skill description in each row.

Click the New Skill button to add a new row at the top of the skills grid.

Select a skill and complete the grade, comment and date acquired column. Note at the date acquired is a six digit date in the format DD/MM/YY.

Click the Save button to process the skills in the EMPSKLS list and update or insert to the employee skills file.

Validation errors will display messages at the top of the page.

## Step 7. Control which Tab is redisplayed

At present the Details WebRoutine has no control over whether the Details or Skills tab is redisplayed. This can easily be achieved by introducing a field that is mapped to set the `tab_index` property.

1. Define a one character field TABINDEX.

```
Define Field(#tabindex) Type(*char) Length(1)
```

2. In the **Details** WebRoutine extend the `web_map` for EMPNO to include TABINDEX as a hidden field.

```
Web_Map For(*both) Fields((#empno *hidden) (#tabindex *hidden))
```

3. In the CASE loop, set the TABINDEX in each When clause, as follows:

```
When = D   #Tabindex := '1'
```

```
When = U   #Tabindex := '1'
```

```
When = N   #Tabindex := '2'
```

```
When = S   #Tabindex := '2'
```

4. Recompile your WAM.
5. Open the **Details** WebRoutine in the *Design* view. Select the Tab Pages weblet and set the `selected_tab_index_field` to TABINDEX. (Select from the dropdown).
6. Ensure the Details tab page is selected and Save your changes
7. Re-test your WAM. When working with the Skills tab, the Skills tab should now be redisplayed after the *New Skill* or the *Save* push button has been used.

Note: This is a limited implementation, using simply the *tab\_index\_field* property. Using a hidden Nav Panel on each tab page, it is possible to return current tab index to a WebRoutine. This allows a design which supports selecting a different employee and always displaying the last tab page used (Details or Skills).

## Step 8. Replace Date Acquired with a Date field (Optional).

In this step you will replace the Date Acquired column in the employee skills list (EMPSKLS) with field STD\_DATEX. This is a Date type field and has a default visualization of a Date Picker weblet.

Note: The default Date Picker visualization for field STD\_DATEX will be automatically implemented when the field is included on the page, or is a column in a simple list weblet.

When used as a column in the Grid weblet the default visualization is not recognised and you will need to add the jQuery UI DatePicker weblet to this column.

1. Change the definition of working list EMPSKLS as shown:

Changes are shown in red.

```
Def_List Name(#empskls) Fields(#SKILCODE #GRADE #COMMENT #std_
Entrys(*max)
```

**Note:** The virtual field DATEACQ will still be required in the list to update the employee skills record. The real field DATEACQR cannot be used for update.

2. In WebRoutine **Details**, when the employee skills list is populated, set up field STD\_DATEX with the value of field DATEACQ:

Changes are shown in red.

```
When (= D)
#tabindex := '1'
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Clr_List Named(#empskls)
Select Fields(#empskls) From_File(pslskl) With_Key(#empno)
#std_datex := #dateacq.asdate( SYSFMT6 )
Add_Entry To_List(#empskls)
Endselect
```

3. When a new list entry is added for insert, set up field STD\_DATEX using current date, and give GRADE a default value (see [Step 9. Change Grade to a Dropdown list \(Optional\)](#)).

Changes are shown in red.

```
When (= N)
#tabindex := '2'
#skl_list := *default
```

```
#std_datex := #std_datex.Now
```

```
#grade := P
```

```
Add_Entry To_List(#empskls) After(*START)
```

```
Message Msgtxt('Complete new skill in top row')
```

4. When employee skills are updated, set up field DATEACQ:

Changes are shown in red.

```
When (= S)
```

```
#tabindex := '2'
```

```
Selectlist Named(#empskls)
```

```
#dateacq := #std_datex.asnumber( DDMMYY )
```

```
If (#dateacqr *NE *zeroes)
```

```
Update Fields(#empskls) In_File(pslskl) With_Key(#empno #skilcode)
```

```
Val_Error(*next)
```

```
If_Status Is_Not(*okay)
```

```
Message Msgtxt('Errors occurred on skills updates')
```

```
Endif
```

```
Else
```

```
Insert Fields(#empskls) To_File(pslskl) Val_Error(*next)
```

```
If_Status Is_Not(*okay)
```

```
Message Msgtxt('Insert failed')
```

```
Else
```

```
Message Msgtxt('Skills for ' + #empno + ' were changed')
```

```
Endif
```

```
Endif
```

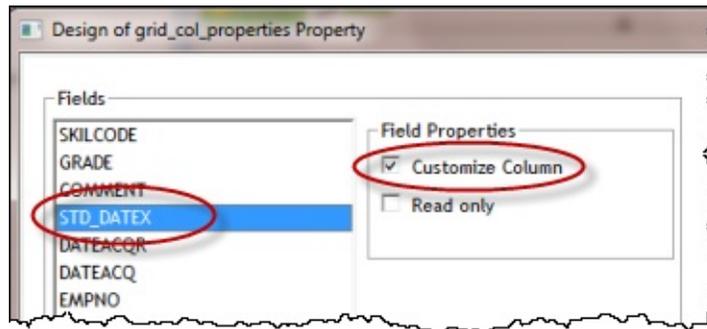
```
Endselect
```

5. At the top of the program, override the column heading for STD\_DATEX, with the following:

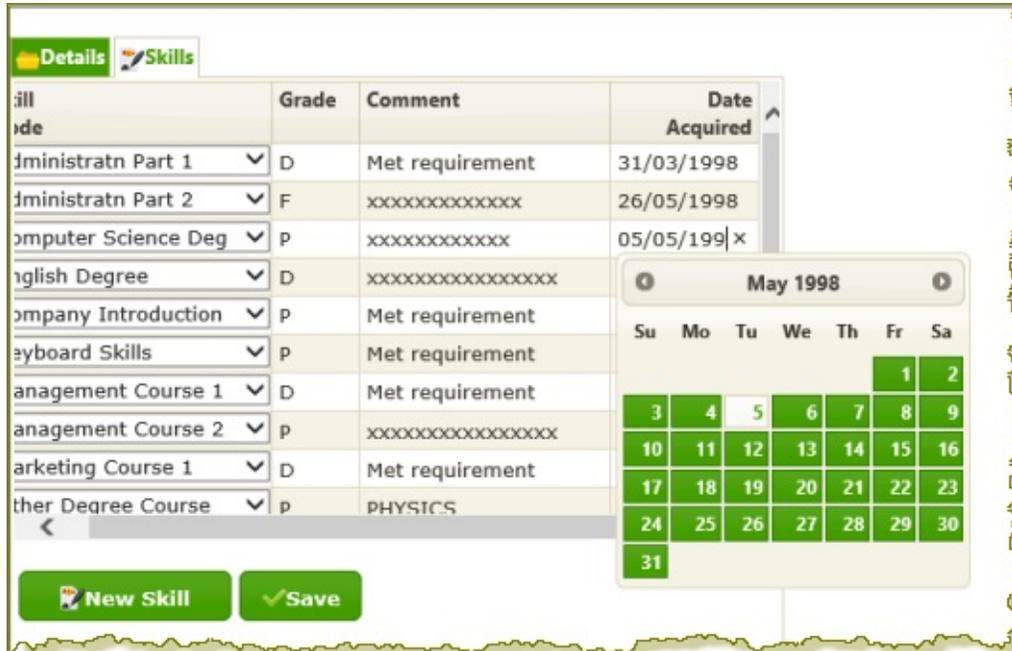
```
Override Field(#std_datex) Colhdg('Date' 'Acquired' '')
```

6. Compile your WAM.
7. Open the **Details** WebRoutine in the *Design* view and select the Skills tab.
  - a. With the Tab Pages weblet selected, change its *content\_width* to **600px**, so that the new date column will be visible in the *Design* view.
  - b. Select the grid. Open the *Design of...* dialog using the ellipsis button for the *grid\_column\_properties* value. Select *Customize Column* for field

STD\_DATEX and click OK.



- c. On the *Weblet Templates* tab, select jQuery UI in the dropdown. Drop a jQuery UI Datepicker weblet into the new Date Acquired column.
  - d. With the jQuery UI Datepicker selected, on the Details tab, change the *dateFormat* to **dd/mm/yyyy** (or use the format suitable for your region).
  - e. Select the Details tab page and save your changes.
8. Retest your WAM. Date Acquired in the Skills grid will now be displayed in dd/mm/ccyy format. Selecting a Date Acquired will display the calendar prompt.



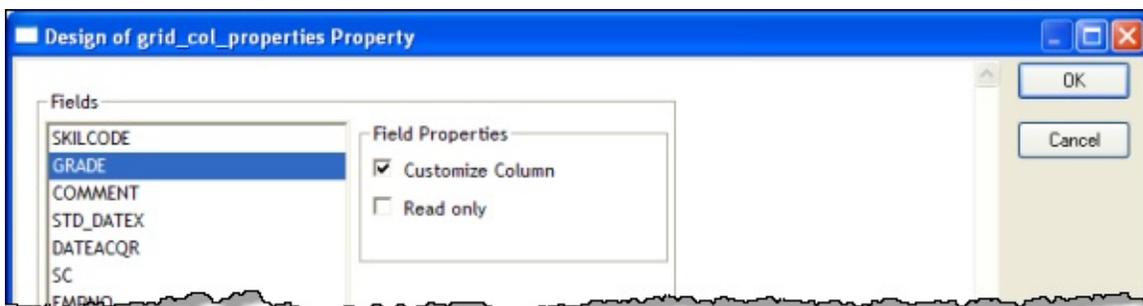
- 9. Change a Date Acquired and Save the changes. Redisplay the employee skills to show the date was updated correctly.

10. Add a new skill. The Date Acquired column should initially contain current date.

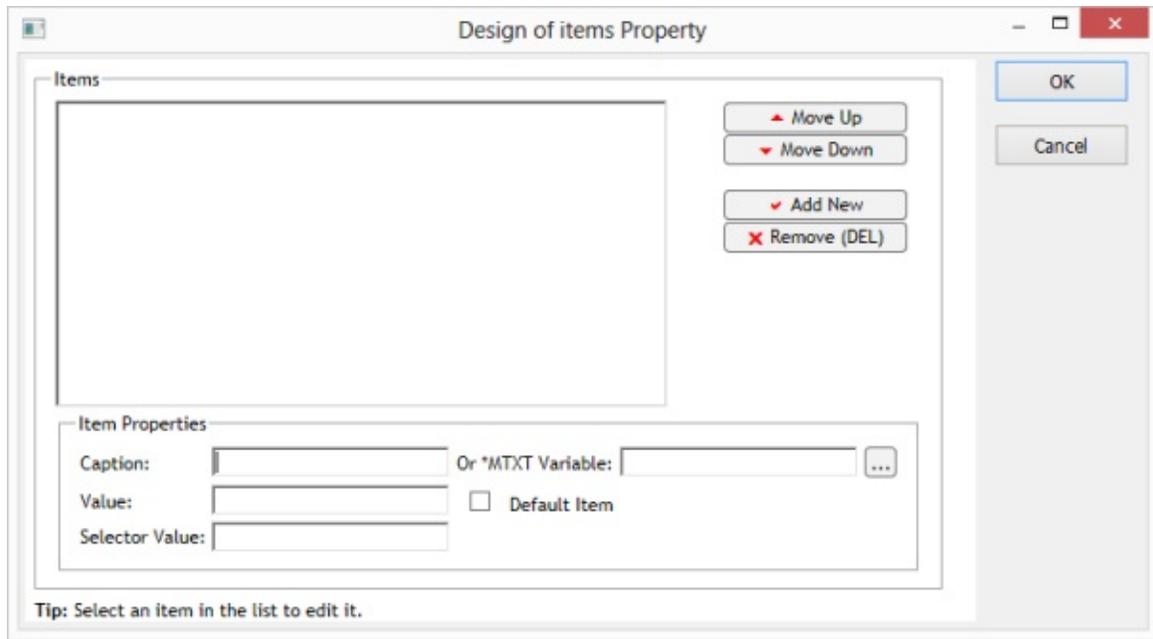
## Step 9. Change Grade to a Dropdown list (Optional)

In this step you will replace the Grade column in the Skills grid with a combo box weblet and set it up with a hard coded list of values.

1. Open the **Details** WebRoutine in the Design view.
  - a. Select the Skills tab.
  - b. Select the Employee Skills Grid.
  - c. On the *Details* tab, select the `grid_col_properties` value and use the Ellipsis button to display the *Design of... Properties* dialog.



- d. Select the `GRADE` field and select the `Customize Column` checkbox. Click *OK* to close the dialog.
  - e. On the *Weblet Templates* tab, select *Standard Weblets* in the dropdown and drag and drop a *combo box weblet* into the `GRADE` column. Increase the width of the *combo box* so that *grade description* can be displayed.
2. Select the *combo box weblet*, to set up its properties.
    - a. Leave the *name* property as `@id` This is correct for identifying a field in a *Grid weblet* (different to a *list*).
    - b. Note that the *value* property is `.` (a dot). This is correct and means *value from current row/column*.
    - c. Select the *items* property value and click the *Ellipsis* button to open the *Design of...* dialog.



d. In the Item Properties group box, set up a list of Captions and Values based on the following table:

<b>Caption</b>	<b>Value</b>
Pass	<b>P</b>
Distinction	<b>D</b>
Merit	<b>M</b>
Fail	<b>F</b>

e. Click *Add New* to add each entry to the list.

f. Click the *OK* button to save the changes.

Your design should look like the following:

Skill Code	Grade	Comment	Date Skl Acquired
ABCDEFGHIJ	Pass	aAbBcCdDeEffGhHiIjJ	12/34/56
ABCDEFGHIJ	Pass	aAbBcCdDeEffGhHiIjJ	12/34/56
ABCDEFGHIJ	Pass	aAbBcCdDeEffGhHiIjJ	12/34/56

New Skill Save

3. Save the WAM.
4. Re-test your WAM. The Employee Skills grid should look like the following, displaying the correct caption for the GRADE fields:

Skill Code	Grade	Comment	Ac
Administratn Part 1	Distinction	Met requirement	25/03/
Administratn Part 2	Fail		02/05/
Communications Degre	Distinction		04/05/
Computer Science Deg	Pass		05/05/
Economics Degree	Pass		01/05/
Company Introduction	Pass	Met requirement	05/02/
Keyboard Skills	Pass	Met requirement	05/02/
Management Course 1	Distinction	Met requirement	15/03/
Marketing Course 1	Distinction	Met requirement	25/03/
Take Shorthand	Pass		03/05/

New Skill Save

5. Change some grades and Save the changes. Redisplay skills for this employee number, to show that grade values were passed into the **Details** WebRoutine.

## Summary

### Important Information

- This WAM combines a number of weblets to build a Windows like interface. This may not be appropriate in all cases, for example this design would not suit a public website or a web site designed for an occasional user.
- The exercise demonstrates how a fairly complex WAM can be built and tested in stages.
- The Dynamic Select Box uses AJAX technology to call a response WebRoutine that refreshes its list of values.
- A Grid column may be customized using a field visualization weblet.
- The Navigation Panel enables one area of the web page to be refreshed (the Navigation panel is an iFrame in HTML terminology).

### Tip & Techniques

- When a new field has been mapped for a WebRoutine, this can be reflected in the web page (actually the XSLT transformation that produces the XHTML) using the WebRoutine Output tab, and dropping the new field onto the page.
- The WebRoutine Output tab reflects the XML, which is always updated by a compile.
- jQuery weblets require lists to be mapped as \*JSON data.

### What I Should Know

- How to combine a number of weblets to construct a Windows-like interface.

## **WAM065 - Controlling List Output**

### **Objectives**

When you output a list to your web page, you should always consider whether there is a need to limit the number of entries that will be displayed.

If you output a large list:

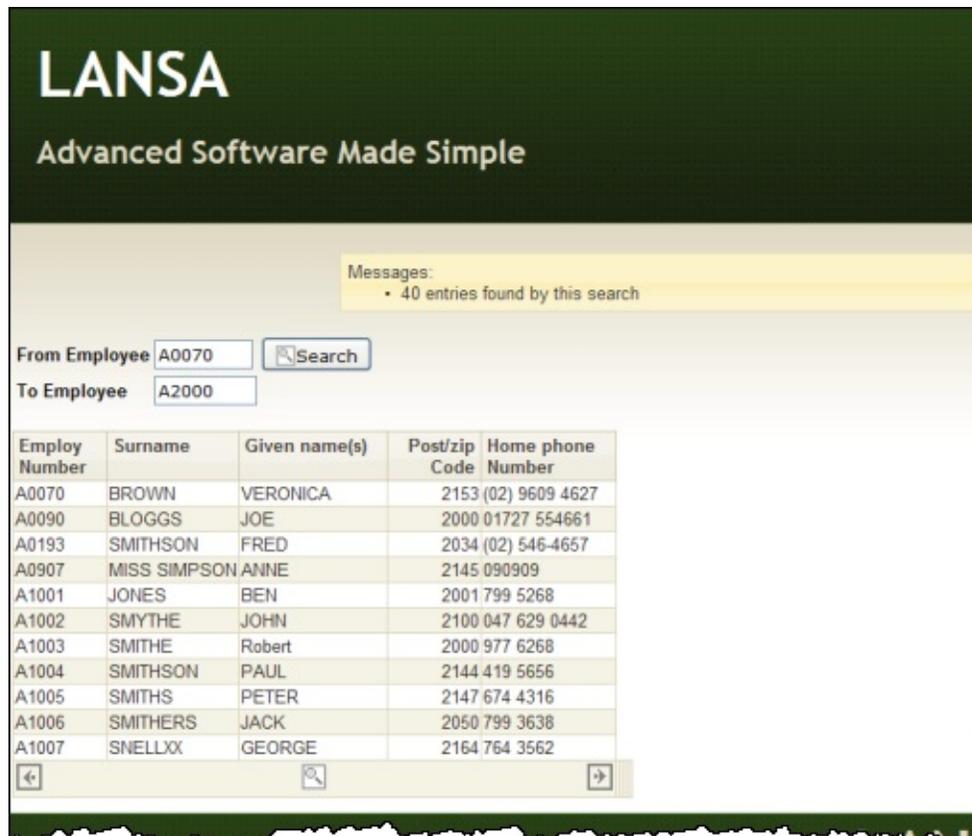
- The application may perform badly.
- The user will need to scroll down the page to find the entries he is interested in, or will need to scroll through the list or grid to find the entries he wants.

If you are an IBM i developer you will be familiar with writing output to sub-files, which the 5250 terminal is then able to scroll through. In the web, if you add 500 entries to the list, they will all be loaded to the page immediately.

This exercise demonstrates how the list paging weblet can be used together with program logic to load a list with one page of entries at a time. This provides one simple technique that could be used.

From your own experience of using web sites you will be familiar with a number of alternative techniques. For example a set of page links, which enable the user to jump to another page of results. These other techniques could also be implemented in a WAM.

When designing applications that may bring back a large set of results, you should always consider as many ways as possible for the user to limit his query to find only the entries he is looking for.



The employee enquiry application:

- Enables employee numbers to be selected using the AutoComplete weblet
- The search displays employees with a fixed page size of 10 entries
- The list\_paging\_weblet enables the user to page forward and backward to display all the employees found by the search

**Note:** The code provided in this exercise can be copied from the WAM tutorials in the VL online guide.

To achieve these objectives you will complete the following:

[Step 1. Create WAM iiiEmpSearch – Employee Search](#)

[Step 2. Add List Paging Images weblet](#)

[Step 3. Add AutoComplete Weblets \(optional\)](#)

[Summary](#)

## Before You Begin

You should have completed all the preceding exercises in this workshop.



Endroutine

EMPFROM and EMPTO will provide search values for employee number. The current page of results will be displayed by the list EMPLIST. A push button will return a STDREENTRY value of S, which will perform the method routine browse.

Ignore the error *Feature name browse is not a member....'*. This routine is defined in the next step.

**Important Note:** This WAM, when completed, will comprise around 130 statements. To save time, much of this code is provided. Make sure you review each section of code as it is added, either at the time you are doing it or later. As with all programming tasks, you should approach WAM development a stage at a time, and test your initial code, before moving on to add additional logic.

6. Create the browse method routine using the following code:

```
Mthroutine Name(browse)
Define_Map For(*input) Class(#empno) Name(#i_empfrom)
Define_Map For(*input) Class(#empno) Name(#i_empto)
Clr_List Named(#emplist)
Select Fields(#emplist) From_File(pslmst) Where((#std_count <= 10) And (#e
Add_Entry To_List(#emplist)
If (#std_count = 1)
#empnof := #EMPNO
Endif
#empnow := #EMPNO
Endselect
#cur_page += 1
Endroutine
```

You will add more logic to this routine in a later step.

Review the browse routine, which:

Accepts two input values for EMPNO.

Clears the working list EMPLIST. Note that STD\_COUNT is the Counter() riable for this list.

Selects records from file PSLMST while STD\_COUNT is less than or equal to and the employee number is less than I\_EMPTO.

Reads the file using a startkey and an endwhere. The read will end when the here() condition is false.

The SELECT reads the next 10 records or stops when the end value for the arch is reached.

Adds entries to list EMPLIST

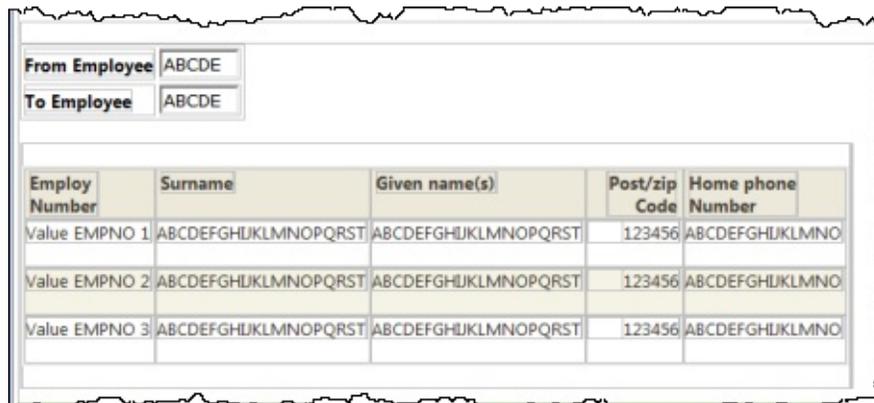
Maintains EMPNOF as the first entry in the current list and EMPNOW as the it entry in the current list

Increments current page (CUR\_PAGE)

7. Add the following as hidden fields in the global web\_map:  
EMPNOF, EMPNOW and CUR\_PAGE.

Note that this web\_map maps the fields for \*both. Their current value will be mapped back into each WebRoutine that is invoked from the web page.

8. Compile your WAM and open the search WebRoutine in the Design view. It should look like the following:



9. Select the employee number input box in the table containing employee numbers, and use the context menu *Table Items / Add Columns...* to add one column to the table.
10. Drop a *Push button with image* into the new top cell.  
Remove the placeholder characters.
11. Select the push button and set up the push button as follows, using the Details tab:

Property	Value
caption	<b>Search</b>

left_relative_image_path	<b>icons/normal/16/zoom_16.png</b>
on_click_wname	<b>search</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: S</b>

12. Set the tab\_index for the push button and employee number fields as follows

<b>Element</b>	<b>Tab_index</b>
EMPFROM	1
EMPTO	2
Push Button	3

13. Save your changes. Your web page should look like the following:

The screenshot shows a web form with two input fields labeled 'From Employee' and 'To Employee', both containing the text 'ABCDE'. To the right of these fields is a search button with a magnifying glass icon and the text 'Search'. Below the form is a table with the following structure:

Employ Number	Surname	Given name(s)	Post/zip Code	Home phone Number
ABCDE	aAbBcCdDeEfGhHiIjJ	aAbBcCdDeEfGhHiIjJ	123456	ABCDEFGHIJKL
ABCDE	aAbBcCdDeEfGhHiIjJ	aAbBcCdDeEfGhHiIjJ	123456	ABCDEFGHIJKL

14. Execute your WAM in the browser. Enter search values for employee number such as A0070 and A2000. Your results should look like the following:

From Employee

 Search

To Employee

Employ Number	Surname	Given name(s)	Post/zip Code	Home phone Number
A0070	BROWN	VERONICA	2153	(02) 9609 4627
A0090	BLOGGS	FRED JOHN ALAN	2220	344-2234454545
A0193	SMITHSON	FRED	2034	(02) 546-4657
A0907	MISS SIMPSON	ANNE	2145	090909
A1001	JONESxx	BEN	2001	799 5268
A1002	SMYTHE	JOHN	2100	047 629 0442
A1003	SMITHE	Robert	2000	977 6268
A1004	SMITHSON	PAUL	2144	419 5656
A1005	SMITHS	PETER	2147	674 4316
A1006	SMITHERS	JACK	2050	799 3638
A1007	SNELL	GEORGE	2164	764 3562

## Step 2. Add List Paging Images weblet

1. In the *Design* view, click anywhere in the employee list and use the context menu, *Table Items / Add Rows...* to add one row to the bottom of this table.  
**Hint:** Click in one of the columns such as Given Name and then use the context menu.
2. Select inside the left hand cell of this new row and use the *Details* tab to set its colspan to 5.
3. Drag a *List paging images* weblet into the new row. Your web page should look like the following:



Employ Number	Surname	Given name(s)	Post/zip Code	Home phone Number
ABCDE	aAbBcCdDeEfFgGhHiIj	aAbBcCdDeEfFgGhHiIj	123456	ABCDEFGHIJKLMNO
ABCDE	aAbBcCdDeEfFgGhHiIj	aAbBcCdDeEfFgGhHiIj	123456	ABCDEFGHIJKLMNO
ABCDE	aAbBcCdDeEfFgGhHiIj	aAbBcCdDeEfFgGhHiIj	123456	ABCDEFGHIJKLMNO

4. Save your changes.
5. Select the list paging images weblet and use the *Details* tab to set up its properties:

Property	Value
prevcondfield	<b>STDPREV</b>
nextcondfield	<b>STDMORE</b>
Rentryfield	<b>STDREENTRY</b>
on_page_wname	<b>page</b>
on_search_wname	<b>search</b>

You will observe that STDPREV, STDMORE and STDREENTRY are already defined. All these fields are defined in the repository. To use this weblet you

must ensure they are mapped appropriately.

The WebRoutine **page** is not yet defined, so you must type in this value.

## 6. Save your changes.

You'll find a detailed definition of all weblets in the *Web Application Modules* guide.

STDPREV and STDMORE are used to show or hide the next and previous images. STDPREV = Y will hide the previous image.

The weblet returns a value in STDREENTRY when the *more* or *previous* image is selected.

- The more image returns **M** in STDREENTRY
- The previous image returns **P** in STDREENTRY
- The search image returns **blank** in STDREENTRY.

## 7. Add STDPREV and STDMORE as hidden fields in the global web\_map, which should now look like the following:

\* Global Maps

```
Web_Map For(*both) Fields((#stdreentry *hidden) (#empnof *hidden)
(#empnow *hidden) (#cur_page *hidden) (#stdmore *hidden) (#stdprev
*hidden))
```

## 8. Create a **page** WebRoutine based on the following:

```
WebRoutine Name(page)
Web_Map For(*both) Fields(#empfrom #empto)
Web_Map For(*output) Fields(#emplist)
Case (#stdreentry)
When (= M)
#com_owner.browse I_Empfrom(#empnow) I_Empto(#empto)
When (= P)
#com_owner.previous I_Empfrom(#empnof) I_Empto(#empfrom)
Endcase
#stdreentry := D
Transfer Toroutine(search)
Endroutine
```

Ignore errors shown because the **previous** method routine does not yet exist.

Review the **page** WebRoutine which:

Maps fields EMPFROM and EMPTO, in and out.

Maps the list EMPLIST out.

EMPNOF and EMPNOW represent the first and last entry in the list EMPLIST.

For **more**, the **browse** method routine is invoked with EMPNOW as the from employee number value.

For **previous**, the **previous** method routine is invoked with EMPNOF as the from employee number value.

9. Create the **previous** method routine based on the following:

```
Mthroutine Name(previous)
Define_Map For(*input) Class(#empno) Name(#i_empfrom)
Define_Map For(*input) Class(#empno) Name(#i_empto)
Clr_List Named(#emplist)
Select Fields(*all) From_File(pslmst) Where((#std_count <= 10) And (#empno
Add_Entry To_List(#emplist) After(*start)
If (#std_count = 1)
#empnow := #EMPNO
Else
#empnof := #EMPNO
Endif
Endselect
#cur_page -= 1
If_Status Is(*beginfile)
Message Msgtxt('No more records')
#stdprev := *blank
Else
#stdprev #stdmore := Y
Endif
Endroutine
```

Review the **previous** method routine, which:

Maps in from employee number and to employee number values.

Clears the list EMPLIST

Selects record from file PSLMST with a startkey, reading backwards, with an dwhere condition.

The read ends when 10 records have been added to the list, or the current

Employee number is equal to or greater than the I\_EMPTO value. This field contains the 'From Employee' value.

EMPNOF and EMPNOW are maintained, allowing for the read being backwards.

Current page number (CUR\_PAGE) is decremented.

Beginning of file is detected with a message.

STDPREV and STDMORE are maintained, and control showing and hiding the *more* and *previous* images.

10. Extend the **browse** method routine to:

Manage STDPREV and STDMORE

Detect end of file

Detect the employee number to value has been reached.

Your code should look like the following. **New** code is shown in red.

```
.....  
Endselect  
#cur_page += 1  
If (#cur_page >= 2)  
#stdprev := Y  
Else  
#stdprev := *blanks  
Endif  
If_Status Is(*endfile)  
Message Msgtxt('No more records')  
#stdmore := *blank  
Endif  
If (#empno >= #empto)  
Add_Entry To_List(#emplist)  
Message Msgtxt('End of Search')  
#stdmore := *blank  
Endif  
Endroutine
```

11. In the **Search** WebRoutine add code to initialize the STDPREV and STDMORE fields.

Your code should look like the following. **New** code is shown in red.

**When (= s)\* Search Button#stdprev := \*blank#stdmore :=  
Y#com\_owner.browse I\_Empfrom(#empfrom) I\_Empto(#empto)**

12. Compile and test your WAM in the browser.
  - a. With search values such as from A0070 to A2000 you should be able to page forward until A2000 is reached (or the next nearest record if A2000 does not exist) and then page back until A0070 is reached.
  - b. Perform a search from A0070 to A0090. The *More* and *Previous* images should not be displayed.
13. Notice that when the Search web page is initially displayed, the list images weblet is shown. To hide this when the page is initially displayed, make the following changes:
  - a. Add the field STD\_COUNT to the global WEB\_MAP as a hidden field
  - b. Open the Search webroutine in the *Design* view and select the std\_list\_images weblet. Define the *hide\_if* property for the std\_list\_images as:  

```
#STD_COUNT = 0
```

This logic must be added in the Xpath expression editor for the *hide\_if* property. Note that the field name must be in upper case.
14. Recompile your WAM and re-test. The list images weblet should be hidden when the Search page is first displayed.

### Step 3. Add AutoComplete Weblets (optional)

This step adds *Autocomplete* weblets for the *To* and *From* employee number fields. You may choose not to complete this step to save time.

At present, to use this enquiry you need to know suitable employee numbers. Adding an *AutoComplete* weblet for the from and to employee fields and a supporting response WebRoutine, will bring back and display a list of matching employee numbers as you type into the weblet.

1. Create the Empno\_Prompt response WebRoutine based on the following code:

```
WebRoutine Name(Empno_Prompt) Response(*JSON)
Web_Map For(*input) Fields(#empno)
Web_Map For(*output) Fields((#emp_dd *json))
Def_List Name(#emp_dd) Fields(#empno #std_code) Counter(#std_count) Type
Clr_List Named(#emp_dd)
Select Fields(#emp_dd) From_File(pslmst) Where(#std_count <= 3) With_Key
#std_code := #empno
Add_Entry To_List(#emp_dd)
Endselect
Endroutine
```

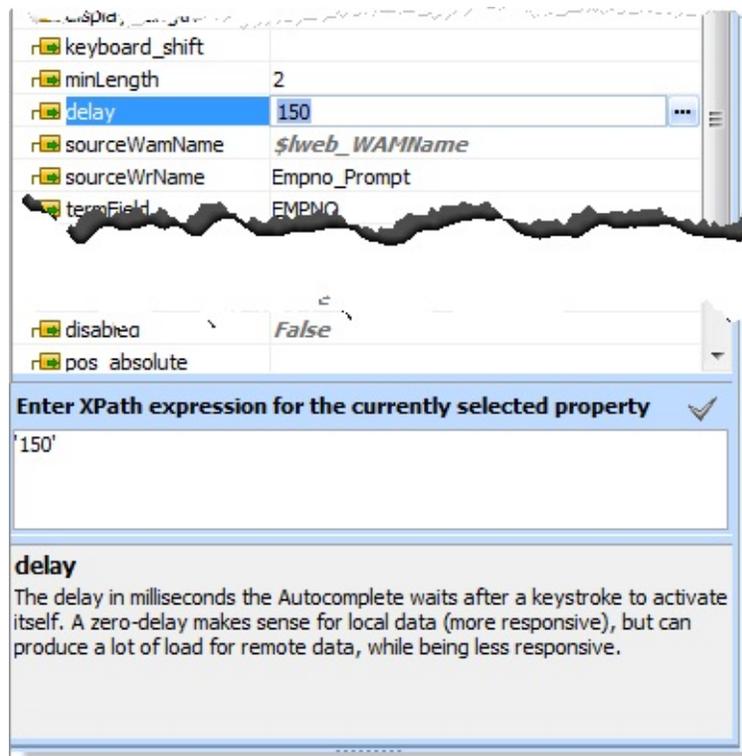
The Empno\_Response WebRoutine:

- Must have the Response(\*JSON) keyword on the WebRoutine statement. This routine will be called by the AutoComplete weblet and returns a small list of employee numbers as JSON data.
  - Maps the field EMPNO for input
  - Maps the list EMP\_DD for output as JSON data.
  - Defines a working list EMP\_DD containing EMPNO. A second field has been added to ensure the fields are recognized by the *Details* tab dialog.
  - Clears the list EMP\_DD
  - Selects up to 3 entries from the file PSLMST using EMPNO as a startkey
  - Adds entries to the list EMP\_DD
  - Returns the list EMP\_DD to the AutoComplete weblet.
2. Compile your WAM.
  3. Open the **search** WebRoutine in the *Design* view.

- Drop an AutoComplete weblet onto to the EMPFROM field and the EMPTO field.
- Set up both AutoComplete weblets as follows:

Property	Value
minLength	2
Delay	150
sourceWrName	<b>Empno_Prompt</b>
termField	<b>EMPNO</b>
listName	<b>EMP_DD</b>
valueField	<b>EMPNO</b>

For more information on all weblets see the *Web Application Modules Guide*. See also the help available for each property from the *Details* tab:



The **minLength** value is the characters to be typed before the WebRoutine is

called

The **delay** value is the number of milliseconds the weblet waits to activate itself after the last keystroke.

The **termField** is the value passed to the WebRoutine defined in **sourceWrName**.

The **listName** is the response list to be displayed as a dropdown list.

The **valueField** is the list value to be displayed.

6. If necessary reduce the width of the AutoComplete input boxes to suit the field EMPNO (up to 5 characters).
7. Save your changes.
8. Execute your WAM in the browser and test the AutoComplete weblet. You should get the following results:

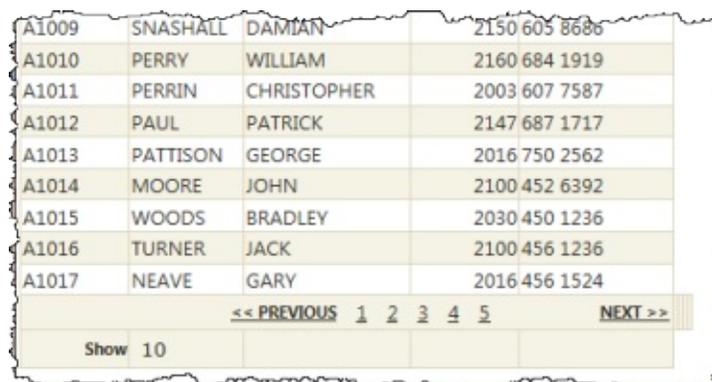


The screenshot shows a weblet interface with a search bar and a dropdown list. The search bar is labeled "From Employee" and contains the text "A0". To the right of the search bar is a green button with a magnifying glass icon and the text "Search". Below the search bar is a dropdown list labeled "To Employee". The dropdown list is currently open and shows four items: "A0070", "A0090", "A0193", and "A0907". The first item, "A0070", is highlighted in green. To the left of the dropdown list, there are two small red asterisks and a small red asterisk below them.

## Summary

### Important Information

- Other techniques you could use to control a large list of results, include showing a list of the page numbers available, as below, and enabling the user to jump to the page required.
- This example also enables the user to change the number of entries per page. One simple way to do this is to add an anchor to the page size value and use this to switch between two values such as 10 and 25.

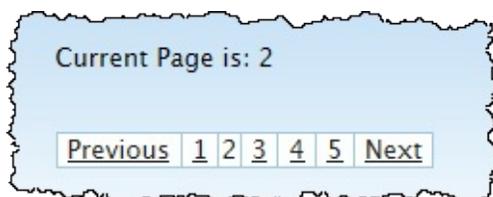


A1009	SNASHALL	DAMIAN	2150 605 8686
A1010	PERRY	WILLIAM	2160 684 1919
A1011	PERRIN	CHRISTOPHER	2003 607 7587
A1012	PAUL	PATRICK	2147 687 1717
A1013	PATTISON	GEORGE	2016 750 2562
A1014	MOORE	JOHN	2100 452 6392
A1015	WOODS	BRADLEY	2030 450 1236
A1016	TURNER	JACK	2100 456 1236
A1017	NEAVE	GARY	2016 456 1524

<< PREVIOUS 1 2 3 4 5 NEXT >>

Show 10

- Another more sophisticated technique uses a working list, displayed as a single row, as in this example. This makes it easy to handle any number of pages and disable the link for the current page.



### Tips & Techniques

- The technique used here will work well on large files.
- Of course your search criteria would likely be much more sophisticated than used here.
- The `SELECT_SQL` statement will enable you to read one or more files rapidly and also implement much more flexible search criteria. See also the free format version of `SELECT_SQL`.
- The *Autocomplete* weblet includes a cache property. If `cache = true`, the selection will be refined locally as you continue to type, rather than going

back to the server. For example the following initial response:

The screenshot shows a web form with two input fields: "From Employee" containing "A0" and "To Employee" containing "A0070". A green "Search" button is to the right. Below the "To Employee" field, a dropdown menu is open, listing employee numbers from A0090 to A1007. The table below the dropdown has columns for "Employ Number", "Surname", "Given name(s)", and "Pos".

Employ Number	Surname	Given name(s)	Pos
A0090			
A0193			
A0907			
A1001			
A1002			
A1003			
A1004			
A1005			
A1006			
A1007			

will be refined to the following as you continue to type into the input box, without making a second call to the server:

The screenshot shows the same web form, but now "From Employee" contains "A01" and "To Employee" contains "A0193". The dropdown menu is still open, but it is now filtered to show only "A0193". The table below the dropdown is partially visible, showing the same column headers: "Employ Number", "Surname", "Given name(s)", and "Pos".

Employ Number	Surname	Given name(s)	Pos
A0193			

## What You Should Know

- How to implement the list paging images weblet. You could also have used the list paging buttons weblet
- How to implement the AutoComplete weblet.

## WAM070 - Hiding Techniques

### Objectives

- To demonstrate different hiding techniques.

In this exercise, you will learn about a number of ways to conditionally hide objects on a page. The conditions will be based on a field value that you will set in the RDML. Some of the hiding techniques will only be available to certain types of objects, while one of the techniques can be applied to anything on the page. This requires the XSL Source to be manually edited.



To achieve these objectives, you will complete the following:

[Step 1. Create a new WAM](#)

[Step 2. Edit the HideMain WebRoutine web page](#)

[Step 3. Apply the Conditional Hides](#)

[Step 4. Test the WAM](#)

[Summary](#)

### Before You Begin

- In order to complete this exercise, you must first complete all the previous Exercises.

## Step 1. Create a new WAM

1. Create a new **WAM**:

Name: **iiiHideTech**

Description: **Hiding Techniques**

Layout Weblet: **iiilay01**

2. In this step, you will add the RDMLX code to demonstrate different hiding techniques.

Use the DEFINE command to define the following work fields.

Field	Type	Length	Default Value
CLASSHIDE	*char	10	*blanks
HIDEIF	*char	10	*blanks
XSLIF	*char	10	*blanks

3. Create a WebRoutine named **HideMain** with a description of '**Hiding Techniques**'.

```
WebRoutine Name(HideMain) Desc('Hiding Techniques')
```

```
Endroutine
```

4. Add a WEB\_MAP for \*both. Include the fields you have just created as well as STDREENTRY. Define all as hidden fields as follows:

```
Web_Map For(*BOTH) Fields((#CLASSHIDE *HIDDEN) (#HIDEIF *HIDD
```

5. Add a CASE loop based on STDREENTRY for three different values A, B and C.

**Note:** Classes are case sensitive (#CLASSHIDE := 'hidden'). 'hidden' must be lower case.

Your code should look like the following:

```
Case Of_Field(#STDREENTRY)
```

```
When Value_Is(= A)
```

```
If Cond(#CLASSHIDE = *BLANKS)
#CLASSHIDE := 'hidden'
Else
#CLASSHIDE := *BLANKS
Endif
When Value_Is(= B)
If Cond(#HIDEIF = *BLANKS)
#HIDEIF := 'HIDE'
Else
#HIDEIF := *BLANKS
Endif
When Value_Is(= C)
If Cond(#XSLIF = *BLANKS)
#XSLIF := 'HIDE'
Else
#XSLIF := *BLANKS
Endif
Endcase
```

The complete WAM source code can be found in [WAM 070. Appendix](#)

## 6. Compile your WAM

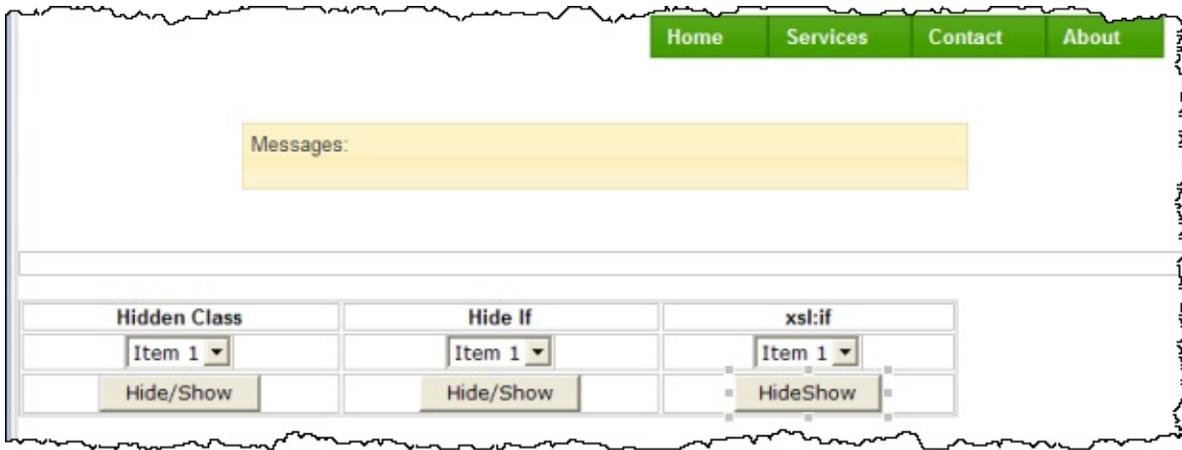
## Step 2. Edit the HideMain WebRoutine web page

In this step, you will set up the web page for the HideMain WebRoutine in the Design view.

1. Open the **HideMain** WebRoutine in the *Design* view.
2. Use the context menu, *Insert HTML / Table* to add a 3 row by 3 column table to the page.
3. Select the table and set the table's width property to **80%**.
4. Set the align property of each individual cell to **center**.
5. Set the *class* property of each cell in the **top row** to **bold**.
6. Add text, to clarify the technique used, to each of the three cells in the top row. The headings should be **Hidden Class**, **hide\_if**, and **xsl:if**.
7. Add a *Combo box* weblet to each of the three cells in the middle row.
8. Add a *Push button* weblet to each of the three cells in the bottom row.
9. Configure the three Hide/Show Push buttons using this table.

	<b>Property</b>	<b>Value</b>
	caption	<b>Hide/Show</b>
	on_click_wrname	<b>HideMain</b>
<b>Table Column 1</b>	submitExtraFields	<b>Field Name: STDREENTRY Literal Value: A</b>
<b>Table Column 2</b>	submitExtraFields	<b>Field Name: STDREENTRY Literal Value: B</b>
<b>Table Column 3</b>	submitExtraFields	<b>Field Name: STDREENTRY Literal Value: C</b>

The *Design* view should appear something like the following:



10. Save your changes.

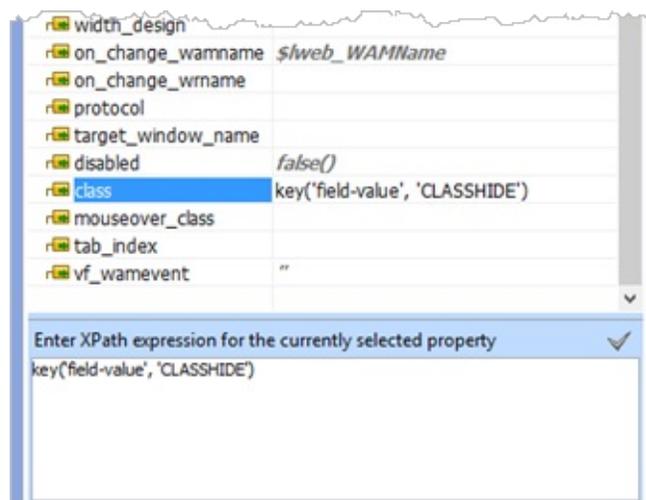
### Step 3. Apply the Conditional Hides

In this step, you will conditional hide the dropdowns using different techniques.

#### 1. Set the hidden class for the first dropdown list.

Set the first **Dropdown's** *class* property to **#CLASSHIDE**. The WAM's RDML conditions this field to contain either \*blanks or 'hidden'.

**Note:** Select the *class* property value and then use the *XPath editor* to enter **#CLASSHIDE**.

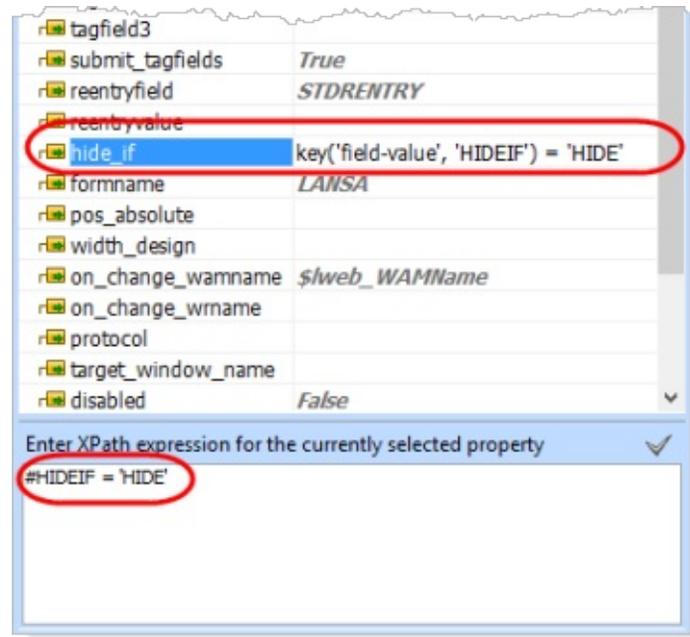


Click the  icon to confirm this change. Note that the XPath editor has added the required XSL expression:

key('field-value', 'CLASSHIDE')

#### 2. Set the hide\_if condition for the second dropdown list.

Select the *hide\_if* property value and use the *XPath expression* window to define the condition **#HIDEIF = 'HIDE'**. Press the  button to confirm this change.



**Note:** The field name must be entered in upper case.

Once again, the field is conditioned in the WAM's RDML, and when the condition specified in the *hide\_if* property is true, the object will be hidden.

### 3. Set the **xsl:if**.

- a. Select the third combo box.
- b. Select the *XSL* tab. The block of code that generates the selected item will be highlighted.

You must enclose this **highlighted code** within the **xsl:if** tags. The syntax for the **xsl:if** is:

```
<xsl:if test="test">
  code to be hidden
</xsl:if>
```

Where *test* is the condition on which the code will be hidden.

- c. Enclose the code for the third dropdown with the **xsl:if** condition. The condition to hide is when #XSLIF is 'HIDE', so *test* will be `key('field-value', 'XSLIF') != 'HIDE'`.

**That is, output the combo box to the page if XSLIF is not equal to 'HIDE'.**

The XSL code should look like the following. New code is shown in red.

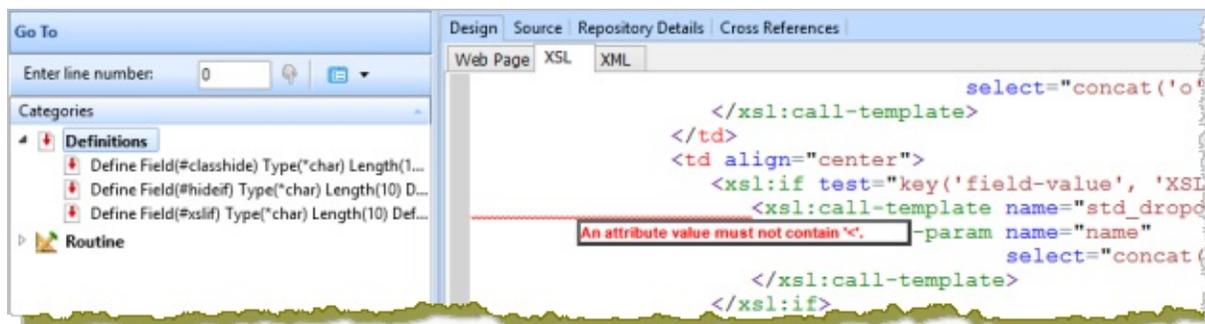
```
<xsl:if test="key('field-value', 'XSLIF') != 'HIDE'">
```

```
<xsl:call-template name="std_dropdown">
  <xsl:with-
param name="name" select="concat('o', position(), '_LANSA_28762')"/>
</xsl:call-template>
</xsl:if>
```

## Using the XSL Editor

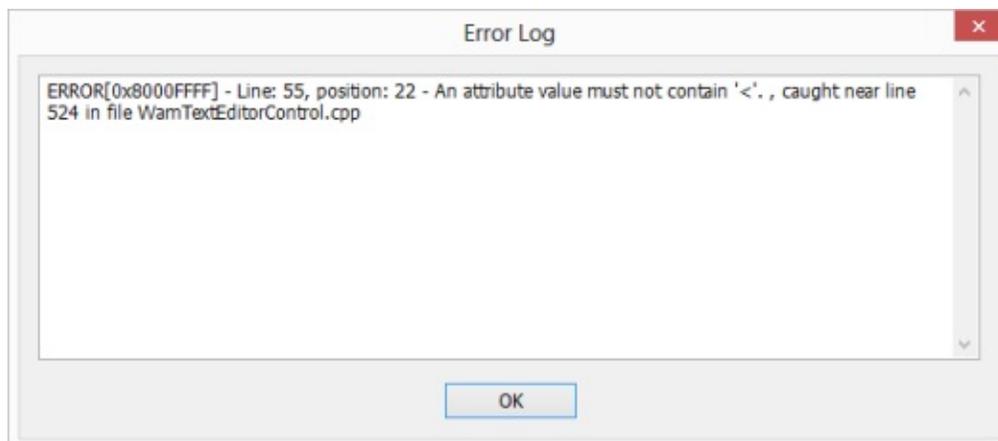
The editor has autocomplete functionality. As you type, press enter to select the prompted code. Once you complete the `<xsl:if` with the `>` character, the end if (`</xsl:if>`) will be generated. Move the "end if" logic after the `</xsl:call-template>`.

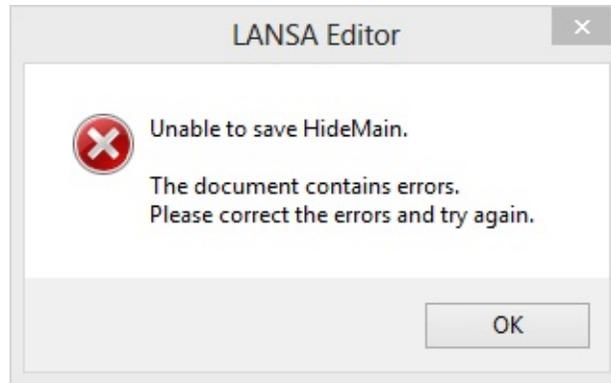
**Note:** If you click on the *Design* tab or try to save your XSL while your xsl changes contain errors, the errors will be reported on the *Go To* tab as shown in this example. In this case, there is a missing double quote at the end of the test condition.



Note that the line in error is highlighted.

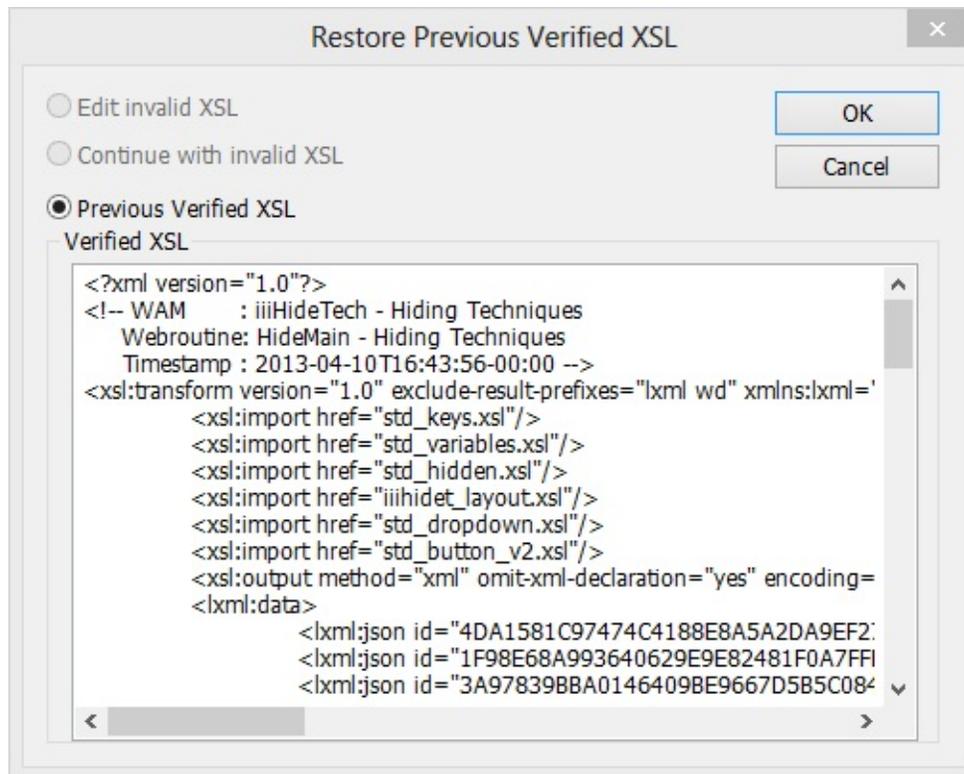
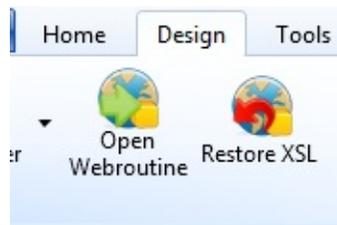
Attempts to save xsl with errors may also show these dialogs:





4. Press OK, to continue editing your XSL. In this case, the error is a missing double quote at the end of the test condition.

On the *Design* ribbon, you can use the Restore XSL dialog to restore the previous verified XSL or continue editing.



5. If necessary, correct any errors in your XSL and save the changes to the

**HideMain** web page.

## Step 4. Test the WAM

In this step, you will test WAM.

1. Run WebRoutine **HideMain** in the browser.
2. Test all three buttons. Pressing each button will hide the object if it is visible, or make the object visible if it is hidden. The buttons will appear to function in the same way, but you know what is going on behind the scenes, and can see the different techniques in action.
3. If your hide / show logic is not working you will need to check both your RDML and on the *Design* tab, the definition of the push buttons and combo boxes. If necessary use debug to resolve any errors.
4. Close the browser.

## **Summary**

### **Important Observations**

- You can apply the hidden class to any object with a class property.
- You can set the hide\_if property to hide weblets.
- When an object does not have a class or hide\_if property, you can still hide it from within the XSL code using an xsl:if.

### **Tips & Techniques**

- Once the objects on the page are set up correctly (conditionally hidden based on a field) you will use the RDML to control the field to hide the object.

### **What I Should Know**

- How to hide object using a hidden class, the hide\_if property, and an xsl:if.

## WAM 070. Appendix

Use the following RDMLX source code to create iiiHideTech in Step 1 of this exercise.

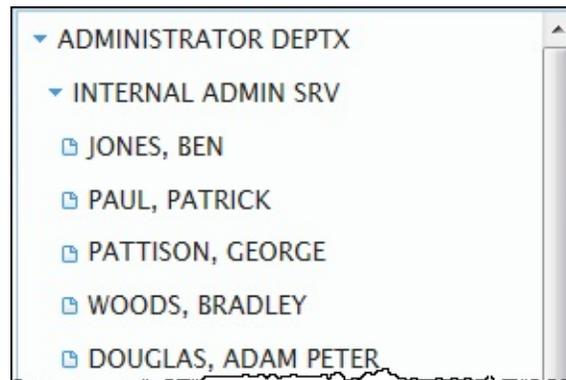
Replace the Layoutweblet() keyword with your common layout name.

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
WebRoutine Name(HideMain)
Web_Map For(*BOTH) Fields((#CLASSHIDE *HIDDEN) (#HIDEIF *HIDD
Define Field(#CLASSHIDE) Type(*char) Length(10)
Define Field(#HIDEIF) Type(*char) Length(10)
Define Field(#XSLIF) Type(*char) Length(10)
Case Of_Field(#STDREENTRY)
When Value_Is(= A)
If Cond(#CLASSHIDE = *BLANKS)
#CLASSHIDE := 'hidden'
Else
#CLASSHIDE := *BLANKS
Endif
When Value_Is(= B)
If Cond(#HIDEIF = *BLANKS)
#HIDEIF := 'HIDE'
Else
#HIDEIF := *BLANKS
Endif
When Value_Is(= C)
If Cond(#XSLIF = *BLANKS)
#XSLIF := 'HIDE'
Else
#XSLIF := *BLANKS
Endif
Endcase
Endroutine
End_Com
```

## WAM075 - Using a Tree View Weblet

### Objectives

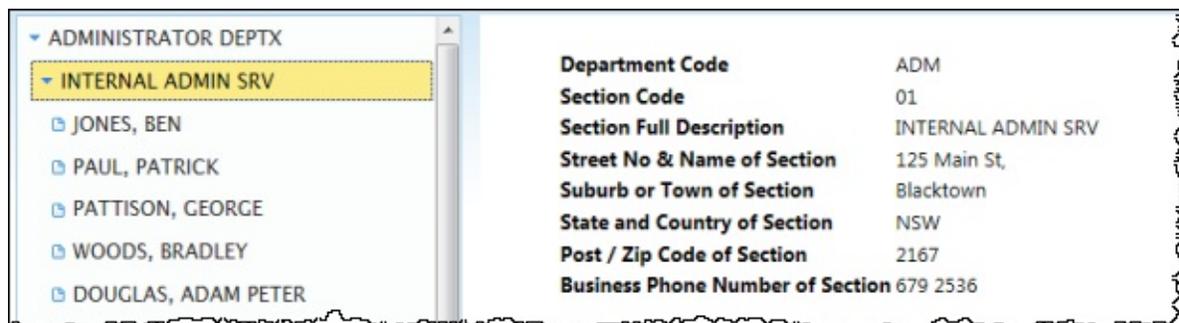
The Tree View weblet provides an expandable collapsible tree. It can be useful as a site navigation system or for visualizing complex hierarchical data.



The Tree View is filled with data from a working list. Similar to the VL Tree View, the source data may be unlevelled, where each entry has an id and specifies the id of its parent, or levelled, where the tree builds itself from a sorted list based on key columns in the list.

When using unlevelled source data, the list can be configured to use AJAX to request child entries from the server when a branch is opened by the user.

This exercise demonstrates how to build an expandable tree view. It will also show how to display detail data when an entry at each level is selected.



To achieve these objectives you will complete the following:

[Step 1. Create WAM iiiTreeView – Using a Tree View Weblet](#)

[Step 2. Make the Tree View Expand](#)

[Step 3. Display Details for a Selected Department](#)

[Step 4. Display Details for Sections and Employees](#)

## Summary

### **Before You Begin**

You should complete all preceding exercises before starting this exercise.

## Step 1. Create WAM iiiTreeView – Using a Tree View Weblet

### Unlevelled List

In an unlevelled list each entry tells the Tree View exactly where it fits in the tree structure by specifying its parent ID. As a minimum, an unlevelled list must contain columns with the following data:

Field	Tree View Property	Description
ID	<b>list_id_field</b>	A unique ID string to identify the entry.
Parent ID	<b>list_parent_id_field</b>	The ID string of the parent entry. An empty string indicates a top level entry.
Caption	<b>list_caption_field</b>	The text to display for the entry.

The Tree View processes the list entries in the supplied order and cannot add an entry to a parent that doesn't exist. It is your responsibility to ensure the list is sorted so that parent items come before their children and items at the same level are in display order.

Additional list fields may be used to control the tree view behaviour and appearance:

- List\_image\_field
- List\_open\_image\_field
- List\_is\_selected\_field
- List\_is\_expanded\_field
- List\_onselect\_wamname\_field
- List\_onslect\_wrname\_field

Refer to [Tree View \(std\\_treeview\\_v2\) Properties](#) for further details.

Your WAM will build an initial list based on the department table (DEPTAB), so that initially the tree will contain a single level. You will later add routines to handle expanding the tree view at each level.

**Note:** The supplied code can be copied from the WAM tutorials in the Visual

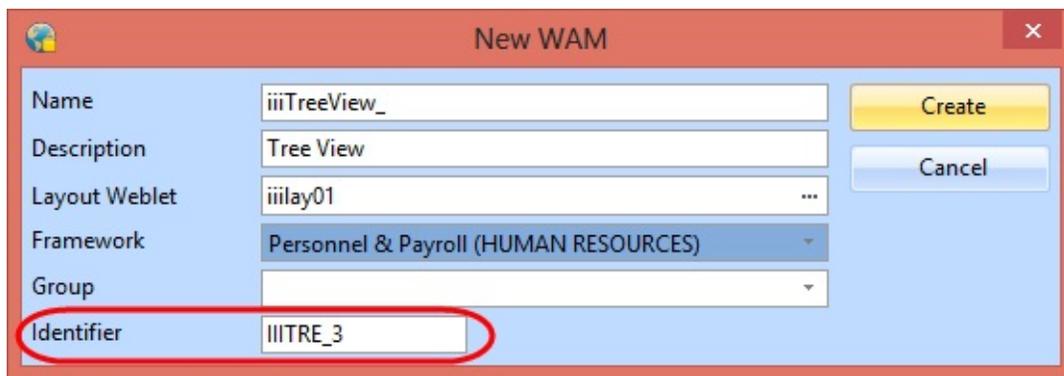
LANSA online guide. Most of the code to support the tree view is provided, but you should ensure that you review it in detail so that you become familiar with the concepts.

1. Create a new WAM:

Name: **iiiTreeView**

Description: **Using a Tree View Weblet**

Layout Weblet: **iiilay01**



Make a note of the Identifier which is assigned to this WAM. You will need to use the Identifier when setting the WAM Name field in the working list which builds the tree view. JavaScript will call the WAM/WebRoutine to display details for the selected level. This requires the WAM Identifier, not the Long Name.

2. Define the fields and lists required to support the tree view

```
Define Field(#listid) Reffld(#std_name)
```

```
Define Field(#listpid) Reffld(#std_name)
```

```
Define Field(#listcapt) Reffld(#std_desc)
```

```
Define Field(#onsubid) Reffld(#std_name)
```

```
Define Field(#onsublvl) Reffld(#std_code)
```

```
Define Field(#haschld) Reffld(#std_flag)
```

```
Define Field(#selwam) Reffld(#std_name)
```

```
Define Field(#selwrn) Reffld(#std_name)
```

```
Define Field(#currid) Reffld(#std_name)
```

```
Def_List Name(#emptree) Fields(#listid #listpid #listcapt #haschld #selwam #
```

```
Def_List Name(#ancestor) Fields(#listid) Type(*Working) Entrys(3)
```

The list named **ancestor** will be explained in a later step.

3. Define a WebRoutine **deptview** which:

Maps for both, the lists EMPTREE and ANCESTOR as JSON data

Clears and builds the list EMPTREE reading all records from the department  
le

Populates the list fields for each entry

**Note:**

The parent id (field LISTPID) is blank for the top level in the tree view.

The "has children" field (HASCHLD) is Y as this is the top level entry.

WAM name and WebRoutine fields in the list define the routine to call if this  
try is selected. The WAM name must be the Identifier which you noted earlier

Your code should look like the following. Substitute your initials for **iii**.

```
WebRoutine Name(deptview)
Web_Map For(*both) Fields((#emptree *json))
Web_Map For(*both) Fields((#ancestor *json))
Clr_List Named(#emptree)
Select Fields(#deptment #deptdesc) From_File(deptab)
#listid := #deptment
#listcapt := #deptdesc
#listpid := *blanks
#haschld := Y
#selwam := iiiTRE_2
#selwrn := DEPDET
Add_Entry To_List(#emptree)
Endselect
Endroutine
```

4. Compile the WAM and open the **deptview** WebRoutine in the *Design* view.
5. Drop a tree view weblet onto the page. Select the tree view weblet and drag the center right hand "handle" to make the tree view wider to allow room for three levels and descriptions to be displayed.
6. Set up the tree view properties as follows:

Property	Value
Listname	<b>EMPTREE</b>
item_image	<b>icons/normal/16/operator_16.png ***</b>

list_caption_field	<b>LISTCAPT</b>
list_id_field	<b>LISTID</b>
list_onselect_wamname	<b>SELWAM</b>
list_onselect_wrname	<b>SELWRN</b>
list_haschildren_field	<b>HASCHLD</b>
list_parent_id_field	<b>LISTPID</b>

\*\*\* select the image using the Ellipsis button. Drill down by selecting the **normal** and **16 folders**.

7. Save your changes and run the WAM in the browser. Your tree view should display description for all departments:



## Step 2. Make the Tree View Expand

In this step you will add a new **treeexpand** WebRoutine to handle expanding departments to add the department's sections and expanding sections to add employees for the selected section.

- The tree view weblet is AJAX enabled and will invoke the expand WebRoutine when an entry in the tree is selected.
- The response WebRoutine invoked by the tree view must have a WebRoutine statement with the keyword Response(\*JSON)
- As before the lists are mapped as JSON data.
- Two additional fields must be mapped into the WebRoutine, one containing the level number being expanded (ONSUBLVL) and the other the id of the selected entry (ONSUBID).
- Depending on the value of field ONSUBLVL , the WebRoutine should add entries from the section table (SECTAB) or the employees file (PSLMST).
- Table DEPTAB is keyed on DEPARTMENT
- Table SECTAB is keyed in DEPARTMENT and SECTION
- File PSLMST is keyed on EMPNO
- Field LISTID contains a unique id for each list entry. Its value must be constructed based on these key relationships.
- LISTID for sections = DEPARTMENT + SECTION
- LISTID for employees = EMPNO
- The parent id field LISTPID must be set using the same values. For example, LISTPID for an employee will be a DEPARTMENT + SECTION value.
- The list ancestors is returned by the tree view weblet and contains 1 to 3 entries, depending on the level being expanded. As its name suggests, an entry contains one field corresponding to the parent of the expanding entry.
- The WebRoutine must retrieve the appropriate entry in the ancestors list to construct the key(s) to read the file and expand the selected entry.
- For section entries, field LISTID contains the concatenated value of DEPARTMENT plus SECTION. The value of SECTION can be extracted from LISTID using the SUBSTRING intrinsic function with a start

position calculated from the actual length of field DEPARTMENT. For example:

```
#std_num := (#department.CurChars + 1)
```

```
.....
```

```
#section := #listid.substring( #std_num )
```

1. Add the following code and then review its logic. Change **iii** to your initials. Ensure that the WAM name contains your WAM Identifier.

```
Webroutine Name(treeexpand) Response(*json)
Web_Map For(*input) Fields((#ancestor *json))
Web_Map For(*output) Fields((#emptree *json))
Web_Map For(*input) Fields(#onsubid #onsublvl)
Clr_List Named(#emptree)
Case (#onsublvl)
* Level 1 expanding – add sections
When (= '1')
#department := #onsubid
Select Fields(#department #section #secdesc) From_File(sectab) With_Key(#depar
#listid := #department + #section
#listcapt := #secdesc
#listpid := #department
#haschld := Y
#selwam := iiiTRE_2
#selwrn := SECDET
Add_Entry To_List(#emptree)
Endselect
* Level 2 expanding – add employees
When (= '2')
Get_Entry Number(1) From_List(#ancestor)
#department := #listid
#std_num := (#department.CurChars + 1)
Get_Entry Number(2) From_List(#ancestor)
#section := #listid.substring( #std_num )
Clr_List Named(#emptree)
Select Fields(#empno #surname #givenname) From_File(pslmst1) With_Key(#c
#listid := #empno
#listcapt := #surname + ', ' + #givenname
#listpid := #department + #section
#haschld := N
```

```

#selwam := iiiTRE_2
#selwrn := EMPDET
Add_Entry To_List(#emptree)
Endselect
Endcase
Endroutine

```

2. Compile your WAM and open the **deptview** WebRoutine in the *Design* view. Select the tree view and complete setting up its properties, as follows:

Property	Value
onexpand_wrname	<b>TREEEXPAND</b>
onsubmit_id_field	<b>ONSUBID</b>
onsubmit_level_field	<b>ONSUBLVL</b>
onsubmit_ancestor_list	<b>ANCESTOR</b>

3. Save your changes and execute your WAM in the browser. Click the *Expand* icon to test the expanding. (Clicking the text to display the details for this level, will be completed in the next step.)

You should now be able to expand a department to add sections belonging to this department and then expand a section, adding employees belonging to this section.



### Step 3. Display Details for a Selected Department

In this step you will add a WebRoutine to display details for the selected department. Details will be displayed on the right hand side of the web page, inside a **Navigation panel**. The Navigation panel weblet enables output to be sent to this area of the page (in HTML terms the Navigation panel is an iFrame), without refreshing the whole page.

1. Create a WebRoutine **DepDet** to display department details.

Field ONSUBID should be mapped into the routine

Fields DEPARTMENT and DEPTDESC should be mapped for output with an tput attribute

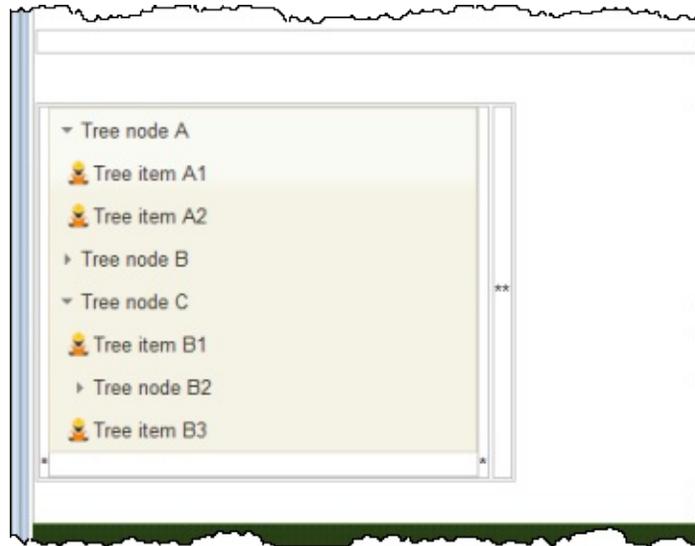
Fetch department fields based on the value of ONSUBID

Your code should look like the following

```
WebRoutine Name(DepDet)
Web_Map For(*input) Fields(#onsubid)
Web_Map For(*output) Fields((#deptment *out) (#deptdesc *out))
#deptment := #onsubid
Fetch Fields(*all) From_File(deptab) With_Key(#deptment)
Endroutine
```

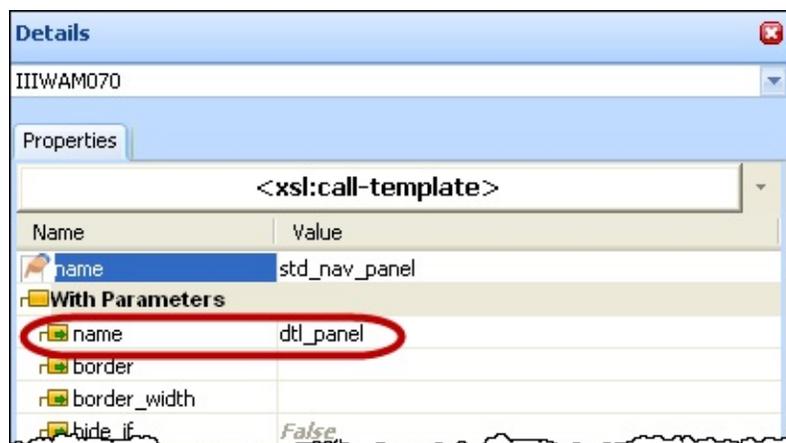
2. Compile your WAM.
3. Open the **DeptView** WebRoutine in the *Design* view. In this step you will add a table with 1 row and 2 columns to the web page, and move the tree view into its left column.
  - a. Position your cursor to the right of the Tree View and press enter to insert a blank line below it.

**Hint:** Select the tree view, move the cursor right using the cursor key and press enter
  - b. Use the context menu to *Insert HTML / Table* with 1 row and two columns, below the tree view.
  - c. Select the tree view and use the context menu to *Cut* it.
  - d. Position the cursor in the left hand column of the table and use the context menu to *Paste* the tree view into it.
  - e. Save your changes. Your web page should now look like the following:



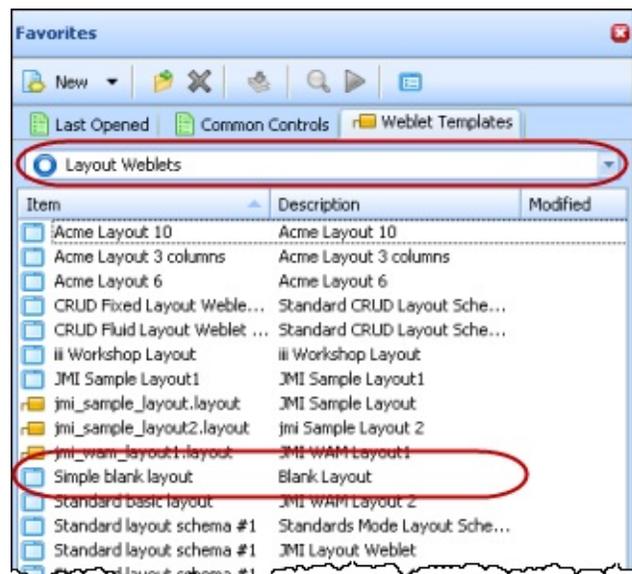
4. In this step you will add a Navigation panel weblet into the right hand cell of the table and set up the tree view to output details for a selected entry to this nav panel.
  - a. Drop a Navigation panel weblet into the right hand cell of the table.
  - b. Select the Navigation panel. Use the *Details* tab to change its name to `dtl_panel`.

**Note:** You are changing the *name* property below the *With Parameters* heading:

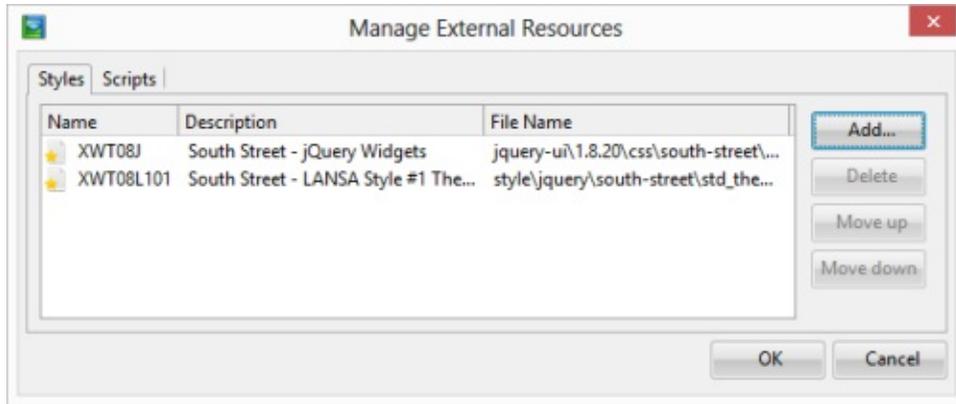


- c. Change the Navigation panel *Size\_panel\_to\_content* to **Yes**.
- d. Change the Navigation panel *border* property to **none**.
- e. Select the tree view. Using the *Details* tab, change its *target\_window\_name* property to `dtl_panel`.

- f. Change the tree view's *node\_text\_click* property to Select. This will select an entry to display details when the entry's **text** is clicked.
  - g. Remove the place holder characters from the table cells.
  - h. With the cursor positioned in the left hand table cell, change its *vAlign* property to top. This will position the tree view at the top of the table cell.
  - i. With the cursor positioned in the right hand table cell, change its *vAlign* property to top.
  - j. Save your changes.
5. In this step you will set up the web page for WebRoutine DepDet, by giving it a blank layout. This WebRoutine will be displayed within the Navigation panel on the **DeptView** web page and therefore does not require a layout.
    - a. Open the **DepDet** WebRoutine in the *Design* view.
    - b. On the *Favorites / Weblet Templates* tab, select Layout Weblets in the top combo box:



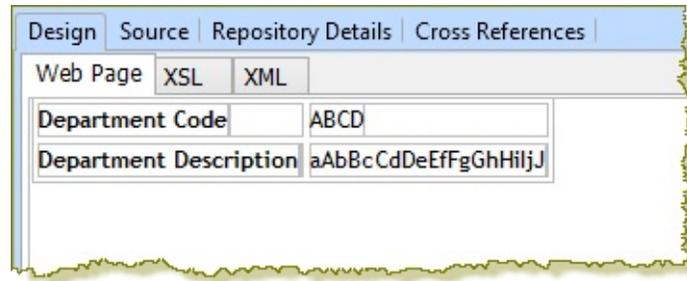
- c. Drag and drop the Simple blank layout onto the web page.
- d. From the *Design* ribbon select External Resources button. In the *Manage External Resources* dialog, add the Style external resources which are required by your chosen common standard layout.



This will add these style sheets into the XSL for this WebRoutines web page and ensure that its contents are consistent with your common layout theme.

**Note:** If necessary, find your common layout (iiilay01) and use the context menu Cross References option to find the external resources being used.

Your design should now look like the following:



6. Save your changes.

## Step 4. Display Details for Sections and Employees

In this step you will complete the WAM by adding WebRoutines to display details for sections and employees.

1. Define a Group\_by SECDATA for fields SECTION, SECDESC, SECADDR1, SECADDR2 SECADDR3, SECPCODE and SECPHBUS. All fields should have an output attribute.
2. Create a new WebRoutine **SecDet**, based on the following:

Map for input field ONSUBID and list ANCESTOR

Map for output the Group\_by SECDATA

Retrieve the first entry from list ANCESTOR and set the value of DEPARTMENT in LISTID

Calculate STD\_NUM based on the actual length of field DEPARTMENT + 1

Assign SECTION by substringing from ONSUBID, starting from STD\_NUM

Fetch department fields with the key DEPARTMENT and SECTION

Your code should look like the following:

```
Webroutine Name(SecDet)
Web_Map For(*input) Fields(#onsubid #ancestor)
Web_Map For(*output) Fields(#secdata)
Get_Entry Number(1) From_List(#ancestor)
#deptment := #listid
#std_num := (#deptment.CurSize + 1)
#section := #onsubid.substring( #std_num )
Fetch Fields(*all) From_File(sectab) With_Key(#deptment #section)
Endroutine
```

3. Compile the WAM.
4. Define a Group\_by **EMPDATA** for fields EMPNO, SURNAME, GIVENAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, PHONEHME, PHONEBUS, DEPARTMENT, SECTION, SALARY, STARTDTE, TERMDATE. All fields should have an output attribute.
5. Define a working list **SKILLS**, for fields SKILCODE, SKILDESC, GRADE, COMMENT, DATEACQ. All fields should have an output attribute.
6. Open the WebRoutine **SecDet** in the *Design* view. Drop the Simple blank

layout onto the page.

7. Use the *Web / Manage External Resources* menu option to give the web page the theme styles to match your common layout.
8. Save your changes.
9. Create a new WebRoutine **EmpDet**.

Map field ONSUBID and list ANCESTOR for input

Map Group\_by and list SKILLS for output

Assign EMPNO to the value of ONSUBID

Fetch employee data from file PSLMST using key EMPNO

Clear the list SKILLS

Build the list SKILLS from file PSLSKL with key EMPNO

Fetch SKILDESC for each employee skill with key SKILCODE

Your code should look like the following:

```
Webroutine Name(EmpDet)
Web_Map For(*input) Fields(#onsubid #ancestor)
Web_Map For(*output) Fields(#empdata #skills)
#empno := #onsubid
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
Clr_List Named(#skills)
Select Fields(#skills) From_File(pslskl) With_Key(#empno)
Fetch Fields(#skildesc) From_File(skltab) With_Key(#SKILCODE)
Add_Entry To_List(#skills)
Endselect
Endroutine
```

10. Compile the WAM.
11. Open the **EmpDet** WebRoutine in the *Design* view. Drop the Simple blank layout onto the web page.
12. On the *Design* ribbon use the *External Resources* button, then *Manage External Resources* dialog to add the Style external resources to match those used in your common layout.
13. Open the **SecDet** WebRoutine in the *Design* view. Drop the Simple blank layout onto the web page.
14. On the *Design* ribbon use the *External Resources* button, then *Manage*

*External Resources* dialog to add the *Style* external resources to match those used in your common layout.

15. Save your changes.
16. Test your WAM by running the **DeptView** WebRoutine in the browser. You should be able to expand the tree view and select a department, a section or an employee to display its details.
17. You probably found that the Navigation panel needed more space to display employee details.
  - a. Open the **DeptView** WebRoutine in the *Design* view
  - b. Select the table containing the tree view and Navigation panel. Do this by selecting a corner of the table. Alternatively, click anywhere inside the table and use the *Outline* tab to locate and select the table.
  - c. Drag the right hand edge of the table to the right.
  - d. Drag the bottom of edge of the table down.
  - e. If you examine the *Style* properties for the table you will see its current size in pixels (for example, width: 800px and height: 560px).
  - f. You can also hover over the center border and drag this to the left to match the size of the tree view weblet.
  - g. If necessary adjust the size of the Navigation panel to use the space now available
  - h. Save your changes
  - i. Re-test your WAM.

## **Summary**

### **Important Information**

- This exercise introduces the basics of building an unlevelled tree, handling its expansion and displaying details for any level. See the *Web Application Modules* guide for more information.
- Other fields in the tree working list may be defined to control open and closed images, and whether an entry is currently selected or expanded.
- The tree view is supported in all LANSAs supported browsers.

### **Tips & Techniques**

- A tree view could be used as an application menu.

### **What You Should Know**

- How to set up and use a tree view weblet for enquiry purposes.

## WAM080 - Session Management

### Objectives

This exercise initially shows how session management operates using a single WAM.

WAM **iiiSessionMng** demonstrates how various WebRoutines within the same WAM can manipulate and share data that is automatically stored on the server. There are a number of key concepts to understand when implementing session management.

- Session management must be enabled at the WAM level. For example `Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01') SessionMng`
- At least one WebRoutine must have an `Onentry()` keyword of `*sessionstatus_none`. This enables the WebRoutine to execute, in order to activate a session.
- Other WebRoutines have a default `Onentry()` keyword of `*sessionstatus_of_wam`. If sessions are enabled for the WAM, then a session must be active for this WebRoutine to run.
- The WAM should contain an event handling routine for `#com_owner.sessioninvalid` that determines what happens when a session is invalid or expired. Usually this will transfer to the "login" WebRoutine to force a session to be established.
- Fields and lists to be stored on the server are defined via a special `web_map`. For example the following defines two working lists that are to be stored as persistent data:

`Web_Map For(*none) Fields(#empsave #empdata) Options(*PERSIST)`

The `For(*none)` keyword value means the fields are not mapped to and from the web page.

- As long as the session is active, each time the WAM starts, the persistent fields and lists are restored.
- Each time the WAM ends, the persistent fields and lists are saved.
- It's important to understand that as each WebRoutine executes, it shares the same persistent fields and lists and their current values as established by the last WebRoutine to execute.

For more details refer to [WAM Session Management](#).

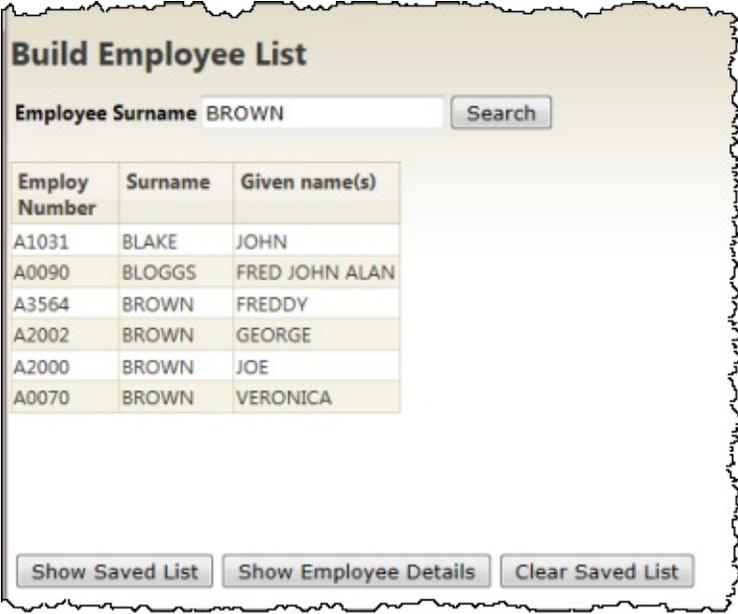
- A second WAM will be developed that shares the session established by iiiSessionMng. This will demonstrate how more than one WAM may share a session and the persistent fields and lists which session management enables.

- WAMs share a session by having a BEGIN\_COM statement that declares Sessionstatus active and has a common Session groupname. For example:

```
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iilay01') Sessio
```

## Description of WAM iiiSessionMng

The **Search** WebRoutine web page will look like the following:



Employ Number	Surname	Given name(s)
A1031	BLAKE	JOHN
A0090	BLOGGS	FRED JOHN ALAN
A3564	BROWN	FREDDY
A2002	BROWN	GEORGE
A2000	BROWN	JOE
A0070	BROWN	VERONICA

- The WebRoutine search will build and display a list of employees, searching the file by surname. Each search adds entries to a working list that is displayed and a second persistent working list which is stored on the server.
- WebRoutine showsave is called by a push button on the search web page. This second routine builds and displays a list from the **saved list**.
- The Clear Saved List pushbutton invokes the **search** WebRoutine to clear the saved list.
- Other functionality will be added in later steps.

To meet these objectives you will complete the following:

[Step 1. Create Session Management 1 WAM](#)

Step 2. Retrieve and Store Employee Details

Step 3. Create Session Management 2 WAM

Step 4. Test the Session Management Application

Summary

## **Before You Begin**

You should complete all preceding exercises in this workshop.

## Step 1. Create Session Management 1 WAM

1. Create a new WAM:

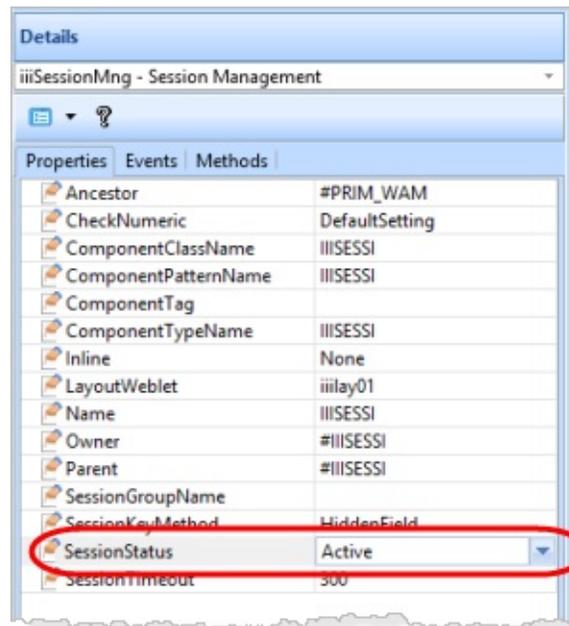
Name: **iiiSessionMng**

Description: **Session Management 1**

Layout Weblet: **iiilay01**

Again, note the **Identifier** assigned to this WAM. You will need this information in a later step. Of course you could always look up the Identifier in the Repository.

2. Press F7 to display the WAM properties. To enable session management, set *SessionStatus* to **Active**.



3. Define the following lists and global web maps:

\* list of employees to be saved on the server

```
Def_List Name(#empsave) Fields(#empno #surname #givenname) Counter(#stdselect)
```

\* latest search list of employees

```
Def_List Name(#empnew) Fields((#empno *out) (#surname *out) (#givenname *out))
```

\* display current saved list of employees.

```
Def_List Name(#empdisp) Fields(#stdselect (#empno *out) (#surname *out) (#givenname *out))
```

\* Map persistent data

```
Web_Map For(*none) Fields(#empsave) Options(*PERSIST)
```

```
* Map common return field from weblets
Web_Map For(*input) Fields((#stdrentry *hidden))
```

Note that working list EMPSAVE is mapped as persistent data.

4. Create three WebRoutines based on the following:

```
* Initialize WebRoutine sets sessionstatus active
WebRoutine Name(init) Onentry(*SESSIONSTATUS_NONE)
#com_owner.sessionstatus := active
Message Msgtxt('Session is now active')
Transfer Toroutine(search)
Endroutine
* Perform search and display results.
WebRoutine Name(search) Desc('Build a list of employees')
Web_Map For(*both) Fields(#surname)
Web_Map For(*output) Fields(#empnew)
Endroutine
* Load and display a list, from the saved list
WebRoutine Name(showsave) Desc('Show saved list of employees')
Web_Map For(*output) Fields(#empdisp)
Endroutine
```

Note that as outlined in the *Objectives*, the init WebRoutine may be executed before a session is active.

5. Add the initial logic to WebRoutine **search**.

when STDRENTRY is S

This should clear the list EMPNEW, which is built by each search.

Ensure the surname is not blank

Build the list EMPLIST by reading the logical file PSLMST2, with a key of rname, with generic(\*yes).

Add entries to both EMPNEW and the saved list EMPSAVE.

Your code should look like the following:

```
Case (#stdrentry)
When (= S)
Clr_List Named(#empnew)
Begincheck
```

```

Valuecheck Field(#surname) With_List(*BLANK) In_List(*ERROR) Not_Inli
Endcheck
Select Fields(#empsave) From_File(pslmst2) With_Key(#surname) Nbr_Keys(
Add_Entry To_List(#empnew)
Add_Entry To_List(#empsave)
Endselect
Endcase

```

6. Add the initial logic to the **showsave** WebRoutine

Output the list EMPDISP which is loaded from the save list EMPSAVE.

When STDREENTRY is L

If the saved list counter value (STD\_COUNT) is greater than 1

- Clear the list EMPDISP
- Read all records from EMPSAVE using SELECTLIST
- Add entries to EMPDISP

Else

- Output message 'Saved list of employees is empty'
- Transfer to **search** WebRoutine.

Your code should look like the following:

```

Case (#stdreentry)
When (= L)
If (#std_count > 1)
Clr_List Named(#empdisp)
Selectlist Named(#empsave)
Add_Entry To_List(#empdisp)
Endselect
Else
Message Msgtxt('Saved employee list is empty')
Transfer Toroutine(search)
Endif
Endcase

```

7. Add an event handling routine for session invalid to transfer to the search **init** WebRoutine. Your code should look like the following:

```

Evroutine Handling(#COM_OWNER.sessioninvalid)
Message Msgtxt('Session Management must be active')
Transfer Toroutine(init)
Endroutine

```

8. Compile the WAM.
9. Open the **Search** WebRoutine in the *Design* view. Add a column to the table containing employee surname. Drop a push button into the new column. Set up the button properties:

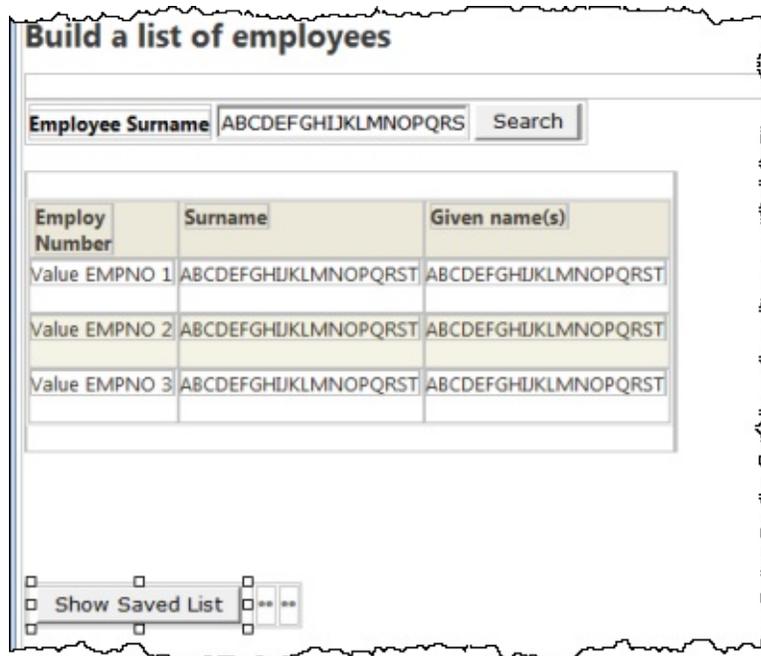
<b>Property</b>	<b>Value</b>
caption	<b>Search</b>
on_click_wrname	<b>Search</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: S</b>

10. Add a few blank lines below the table containing the employees list and insert a table with 1 row and 3 columns. Drop a push button into the left hand column and set up the button properties:

<b>Property</b>	<b>Value</b>
Caption	<b>Show Saved List</b>
On_click_wrname	<b>Showsave</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: L</b>

Adjust the width of the push button to show the caption as a single line.  
 Save your changes.

Your page should look like the following:



11. Open the **showsave** WebRoutine in the *Design* view. Select the list, move the cursor right and press enter to create a blank line below the list. Drop a push button below the list and set up its properties:

Property	Value
Caption	<b>Return</b>
On_click_wrname	<b>Search</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: M</b>

Your web page should look like the following:

**Show saved list of employees**

Std *WEBEVENT template field	Employ Number	Surname	Given name(s)
A	Value EMPNO 1	ABCDEFGHIJKLMNOPQRST	ABCDEFGHIJKLMNOPQRST
A	Value EMPNO 2	ABCDEFGHIJKLMNOPQRST	ABCDEFGHIJKLMNOPQRST
A	Value EMPNO 3	ABCDEFGHIJKLMNOPQRST	ABCDEFGHIJKLMNOPQRST

Return

12. Save your changes.

13. Execute your WAM in the browser by running any WebRoutine. Control will pass to the **init** WebRoutine which will enable session management, and transfer to the **search** WebRoutine.

You should be able to see the following results:

A list of employees based on a search value such as B or S.

Display the current saved list in WebRoutine **showpage**.

Return to the WebRoutine **search**.

Perform another search and display the result of both searches in the **showpage** WebRoutine.

Restart the WAM from the **init** WebRoutine and immediately display the saved t.

Q. Why is the list empty?

A. The persistent data is only restored if the session is already active. Running WebRoutine **init** starts a new session.

## Step 2. Retrieve and Store Employee Details

- In this step you will extend the WebRoutine **showsave**, to fetch employee details. This will be invoked via a clickable image weblet in the list EMPDISP displayed by WebRoutine **ShowSave**.
- An Employee's details will be stored in a new working list EMPDATA that will hold one entry and will also be mapped as persistent data.
- A new WebRoutine **showemp** will display the stored employee details.
- The **search** WebRoutine will be extended to handle clearing the lists EMPSAVE and EMPDATA.

1. Define a working list EMPDATA for employee details:

```
Def_List Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #AD
```

2. Define a group by to display employee fields

```
Group_By Name(#empgrp) Fields((#EMPNO *out) (#SURNAME *out) (#GI
```

3. Extend the web\_map for persistent data to include list EMPDATA.

```
Web_Map For(*none) Fields(#empsave #empdata) Options(*PERSIST)
```

4. Extend case loop in the search WebRoutine to clear the saved lists.

```
When (= C)  
Clr_List Named(#empsave)  
Clr_List Named(#empdata)  
Message Msgtxt('Saved employee list was cleared')
```

5. Extend the case loop in the **showsave** WebRoutine to fetch employee data and add an entry to the list EMPDATA.

```
When (= D)  
Clr_List Named(#empdata)  
Fetch Fields(#empgrp) From_File(pslmst) With_Key(#empno) Val_Error(*nex  
If_Status Is(*okay)  
Add_Entry To_List(#empdata)  
Message Msgtxt('Employee details saved')  
Endif  
Transfer Toroutine(search)
```

- In the **showsave** WebRoutine, add a web\_map for input for field EMPNO. This value will be passed in for the selected row, by the clickable image.

```
Web_Map For(*input) Fields(#empno)
```

- Create a new WebRoutine **showemp** to retrieve the one entry from the list EMPDATA or transfer to the **search** WebRoutine.

```
WebRoutine Name(showemp) Desc('Show Saved Employee')
Web_Map For(*output) Fields(#empgrp)
If (#listcount = 1)
Get_Entry Number(1) From_List(#empdata)
Else
Message Msgtxt('Employee details not available')
Transfer Toroutine(search)
Endif
Endroutine
```

- Compile your WAM.
- Open the **search** WebRoutine in the Design view.
- Add a push button into the table at the bottom of the page. Set up the button properties:

Property	Value
caption	<b>Show Employee Details</b>
on_click_wrname	<b>showemp</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: D</b>

- Add a third push button into the table at the bottom of the page and setup the button properties:

Property	Value
	<b>Clear Saved Lists</b>

caption	
on_click_wname	<b>search</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: C</b>

12. Remove the place holder characters from this table.
13. Save your changes.
14. Open the **showsave** WebRoutine in the Design view.
15. Select and delete the column heading for the first column.
16. Drop a clickable image into the first column (field STDSELECT). Set up the clickable image properties:

<b>Property</b>	<b>Value</b>
currentrowhfield	<b>EMPNO</b>
Currentrownumvalue	<b>\$EMPNO</b>
Reentryvalue	<b>D</b>
On_click_wname	<b>showsave</b>

17. Save your changes.
18. Open the **showemp** WebRoutine in the *Design* view. Drop a push button onto the page below the list. Set up the button properties:

<b>Property</b>	<b>Value</b>
Caption	<b>Return to Search</b>
On_click_wname	<b>search</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: M</b>

19. Save your changes.

20. Test your WAM.

a. You should now get the following results:

Retrieve and display employee search results

Display saved list of employees

Select an employee in the saved list using the clickable image

Display selected employee and return to the search page

Display the saved employee data

Clear both saved employee list and employee data

b. Start your WAM from any WebRoutine except WebRoutine init. You should be transferred to the **init** WebRoutine to establish a session, and then transferred to the **search** WebRoutine, displaying suitable messages.

21. In your WAM source, use F7 to display WAM properties on the *Details* tab. Change the *Session Timeout* from its default value (**300** seconds) to **10** and recompile your WAM.

22. Test your WAM.

Perform a search and display the saved list. This should be displayed. However, due to the very short time out (10 seconds) by the time you return to the search web page, the WAM will have timed out, giving appropriate messages. If you then immediately display the saved employee list, there will be no entries.

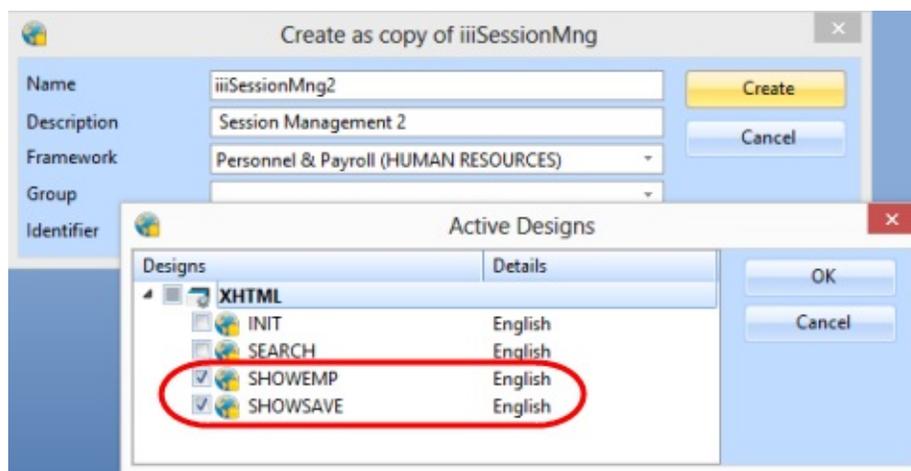
**Note:** The session timeout value is the "wait time" for the response from the client (the browser). So with a session timeout of 10 seconds, whenever you delay for more than 10 seconds, the session will time out. If on the other hand you keep sending requests to the server within the 10 second time out window, the session will never time out.

**Persistent data will only be retrieved if the session is active. If a new session is established, due to time out, there is currently no persistent data for that session.**

### Step 3. Create Session Management 2 WAM

In this step you will create a second WAM and enable session management in both WAMs so that they share a common Session GroupName. This makes both WAMs part of the same session. The new WAM will contain the WebRoutines showsave and showemp and the session invalid event handling routine.

1. Create WAM `iiiSessionMng2` by copying WAM `iiiSessionMng`. Perform the copy step as follows:
  - a. Select the WAM `iiiSessionMng` on the Favorites / Last Opened tab and *Copy* it using the context menu.
  - b. In the *Active Designs* dialog, select only `SHOWEMP` and `SHOWSAVE`.



2. Delete the WebRoutines **INIT** and **SEARCH** from the new WAM.
3. Change four transfer commands to specify the WAM Identifier as well as the WebRoutine name, for example:  
Transfer Toroutine(#iiiSES\_2.search)
4. Press F7 to display the WAM properties on the *Details* tab. Change the Session GroupName to `IIIGROUP`.  
Set the Session Timeout value to 300.
5. Compile WAM **iiiSessionMng2**.
6. Open the `showsave` WebRoutine in the *Design* view and give the *Return* push button an `on_click_wamname` property of `iiiSessionMng`.
7. Open the **showemp** WebRoutine in the *Design* view and give the *Return to Search* push button an `on_click_wamname` property of `iiiSessionMng`.

8. Save your changes.
9. If necessary open WAM **iiiSessionMng** in the editor.
10. Open the search WebRoutine in the *Design* view. Make the following changes:

<b>Push Button</b>	<b>Property</b>	<b>Value</b>
Show Saved List	on_click_wamname	<b>iiiSessionMng2</b>
Show Employee Details	on_click_wamname	<b>iiiSessionMng</b>

- 11 Save your changes.
12. Press F7 to display the WAM properties on the Details tab. Change the Session GroupName to IIIGROUP.  
Ensure the Session Timeout value to 300.
13. Compile WAM **iiiSessionMng**.

## **Step 4. Test the Session Management Application**

You can now test the application by executing any WebRoutine in either of the two WAMs. The session invalid event handling routine will transfer to the init WebRoutine in iiiSessionMng, which will make the session active.

- The application should behave exactly the same as before. The show saved list, retrieve employee details and display saved employee details logic is now performed by WAM iiiSessionMng2.

## Summary

### Important Observations

- Session Management and persistent data provide powerful and easy to use functionality.
- Applications that handle sensitive data can be made to time out after a given wait time
- Your application could establish a common session when the user logs in.
- Common data that is established at log in could be made available to all WAMs via persistent data.
- Data required in a later step can be stored as persistent data.
- Persistent data involves additional database I/Os. Consider carefully how much data needs to be stored and restored as persistent data.

### Tips & Techniques

- Persistent data is secure because it is stored on the server.
- Persistent data is secure and avoids the need to map data as hidden in and out of the web page which may introduce opportunities to "hack" your application.
- Persistent data is only available for the duration of the session. If you need to make the data permanent, then your application must store it in a database.
- Before implementing Session Management in your own applications, see the Web Application Modules guide for more detailed information on Session Management

### What You Should Know

- How to implement session management.

## **WAM085 - Enhancing the User Interface**

### **Objectives**

- To demonstrate a number of weblets which enhance the user interface provided by your web page.
- To illustrate standard field visualization weblets, which are automatically added for certain field types.
- To demonstrate some of the properties of these field visualization weblets

To meet these objectives you will complete the following steps:

[Step 1. Create Repository Field Definitions](#)

[Step 2. Create Employee Number AutoComplete WAM](#)

[Step 3. Create WAM iiiEnhancedUI – Enhancing the Interface](#)

[Step 4. Define Work Fields and Lists](#)

[Step 5. Complete WAM RDMLX](#)

[Step 6. Design the web pages](#)

[Step 7. Test the WAM](#)

[Step 8. Improve the ShowPage Page Design](#)

[Step 9. Insert a fieldset around each table](#)

[Summary](#)

### **Before You Begin**

You should first complete all preceding exercises in this workshop.

Review the following Field Visualations description.

### **Field Visualizations**

Fields defined in the Repository may also have *Field Visualizations* defined.

- These visualizations are able to define the field's appearance in Windows desktop applications and / or in web applications built using WAMs.
- Field visualizations for the web are weblets.
- Fields may have a number of visualizations for Windows and web.
- One field visualization, must be defined as the default visualization, which will be the appearance option that is automatically selected.
- For web, just one weblet visualization should be defined, since at design time there is no way to switch between visualizations, except by deleting

one and adding another.

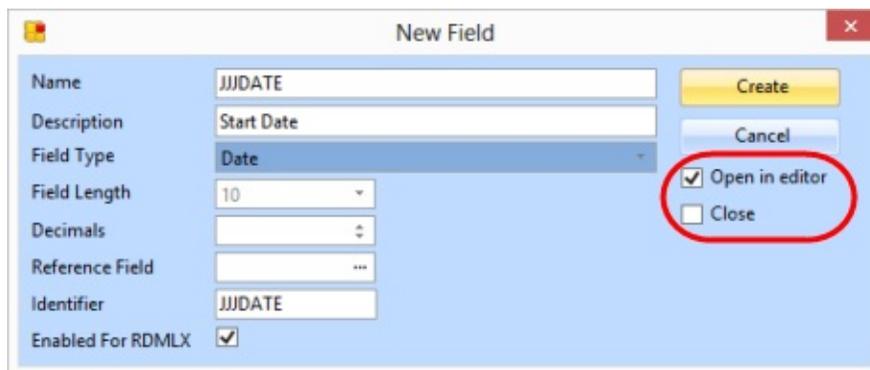
- Fields may have a *static picklist* defined which defines a set of descriptions (or captions) and values. Weblets such as a combo box or radio button group will use this picklist to populate the selection options in the web page.
- Field visualizations are compiled into the web page at design time. A change to the definition of a field's visualization in the *Repository*, for example changing a radio group to a combo box, or adding or removing entries in the picklist, can only be reflected in the web page by removing that field from the web page and re-adding it. It follows that field visualizations defined in the *Repository* with a static picklist, are only useful for very static data that is unlikely to ever change, such as "male or female" or "married, single or divorced".
- Note also that field types such as Date, DateTime and String will probably always benefit from having a weblet such as datepicker, so their automatic visualization definitions in the *Repository* are always useful.
- There are other field visualization options that only apply to Windows desktop applications, that are not described here. For more information, refer to [Field Visualization Development](#) in the *Visual LANSAs Developer Guide*.

## Step 1. Create Repository Field Definitions

1. Create new field definitions in the Repository:

Name	Description	Type	Length	Input Attribute
iiiDATE	Start Date	Date	10	
iiiDTETME	Last Updated	DateTime	25	
iiiNOTE	Comments	String	512	LC
iiiGENDER	Gender	Alphanumeric	1	
iiiTIME	Create Time	Time	8	
iiiCUREMP	Current Employee	Alphanumeric	1	

Use the New Field dialog, to open each field in the Editor and for the dialog to remain open. Fields will be automatically flagged as RDMLX when a type such as Date is selected:

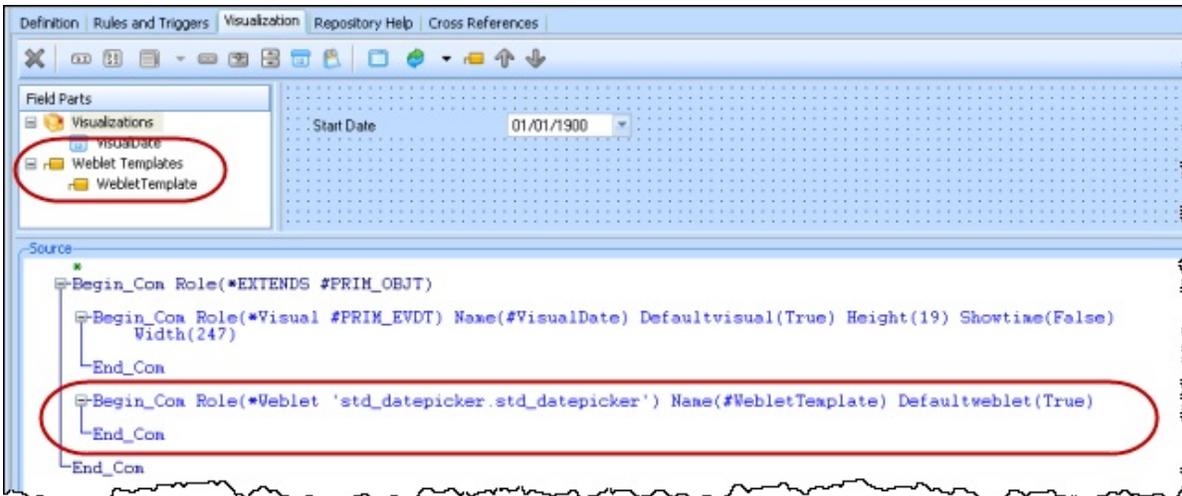


The screenshot shows a 'New Field' dialog box with the following fields and values:

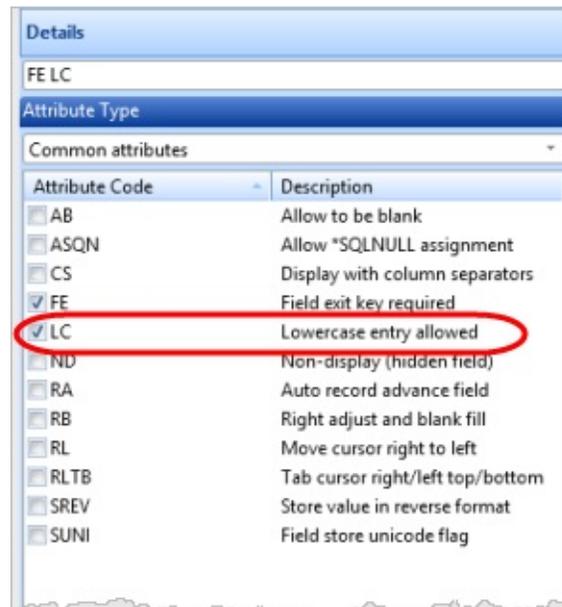
- Name: JJDATE
- Description: Start Date
- Field Type: Date
- Field Length: 10
- Decimals: (empty)
- Reference Field: (empty)
- Identifier: JJDATE
- Enabled For RDMLX:

The 'Open in editor' checkbox is checked and circled in red.

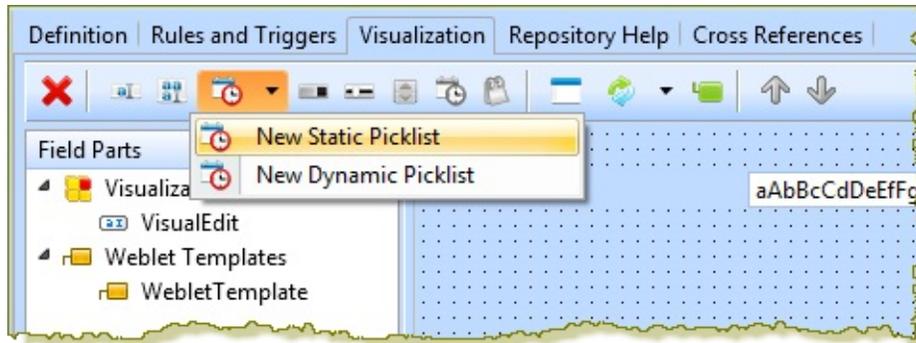
2. Review the *Field Visualization* for the fields iiiDATE, iiiDTETME, iiiTIME and iiiNOTE. They each have been given a suitable weblet visualization.



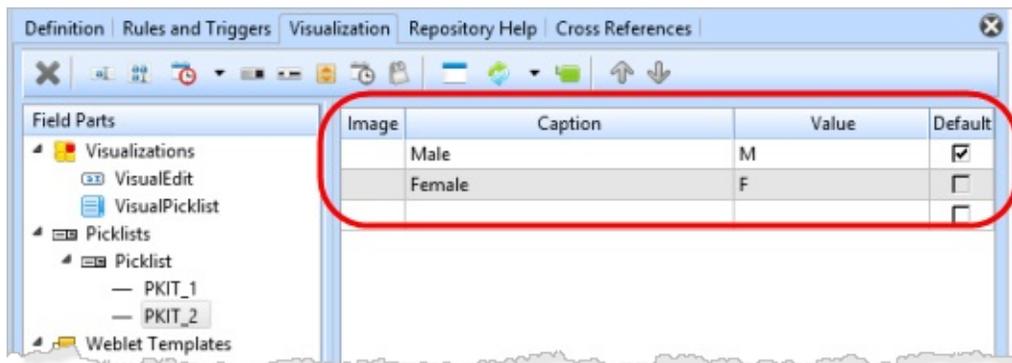
3. Ensure that the *Input* attribute of field iiiNOTE allows lower case:



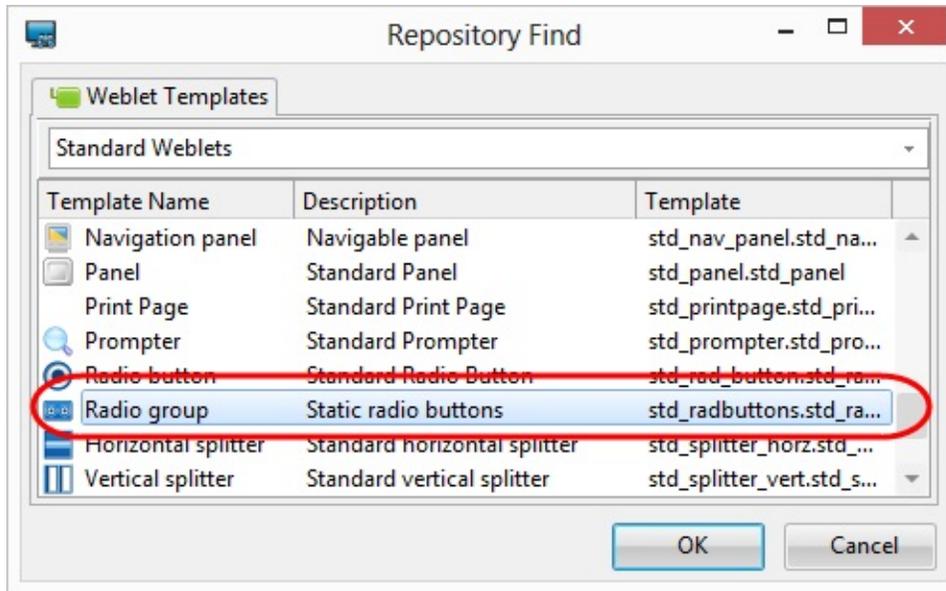
4. Switch to the *Field Visualization* definition for field iiiGENDER. Add a Static Picklist using the toolbar button:



5. Define the static picklist with two entries as shown:



6. Open the *Repository Find* dialog using the  *insert Weblet Template* toolbar button.
- If necessary, in the *Repository Find* dialog, select *Standard Weblets* in the dropdown.
  - Add a *Radio Group* weblet.



7. Save the field definition.

8. Switch to the *Visualization* tab for field iiiCUREMP. Use the  toolbar button to add a Checkbox weblet visualization.

The Checkbox weblet will return, by default, values of Y and N which are correct values for field iiiCUREMP. In this case a static picklist is not required.

9. Save the field definition.

## Step 2. Create Employee Number AutoComplete WAM

This step illustrates how you can create a small response WAM to service any web page that uses an AutoComplete weblet for field employee number (EMPNO).

1. Create a new WAM:

Name: **iiiEmpAutoCmpl**

Description: **Employee Number AutoComplete**

Layout Weblet: **iiilay01**

2. Copy the following WebRoutine from WAM **iiiEmpSearch**.

```
WebRoutine Name(Empno_Prompt) Response(*JSON)
Web_Map For(*input) Fields(#empno)
Web_Map For(*output) Fields((#emp_dd *json))
Def_List Name(#emp_dd) Fields(#empno #std_code) Counter(#std_count) Type
Clr_List Named(#emp_dd)
Select Fields(#emp_dd) From_File(pslmst) Where(#std_count <= 3) With_Key
#std_code := #empno
Add_Entry To_List(#emp_dd)
Endselect
Endroutine
```

3. Compile the new WAM.

### Step 3. Create WAM **iiiEnhancedUI** – Enhancing the Interface

This WAM will use the new fields that you have just created. It will also build on the previous exercise, by enabling session management and saving a working list of each set of data entered.

This WAM will operate as follows:

- Executes WebRoutine **begin** to start session management and transfers to the **select** WebRoutine.
- The **select** WebRoutine requests input of an employee number. The **showpage** WebRoutine is invoked via a push button.
- The **select** WebRoutine also supports a push button to clear the saved list (SAVLIST).
- The **select** WebRoutine includes a push button to invoke the **showlist** WebRoutine.
- The **showpage** WebRoutine fetches employee data and displays all fields (except employee number) for input
- A Save button invokes the **showpage** WebRoutine to save the data to the working list SAVLIST and transfers back to the **select** WebRoutine.
- The **showlist** WebRoutine loads the saved list (SAVLIST) into the list EMPLIST and displays this list.

1. Create a new WAM:

Name: **iiiEnhancedUI**

Description: **Enhancing the Interface**

Layout weblet: **iiilay01**

2. Create the following four WebRoutines:

\* Start session and transfer to select

Webroutine Name(begin) Onentry(\*sessionstatus\_none)

Endroutine

\* Request an employee number

Webroutine Name(select) Desc('Select an Employee')

Endroutine

\* Fetch employee and accept input

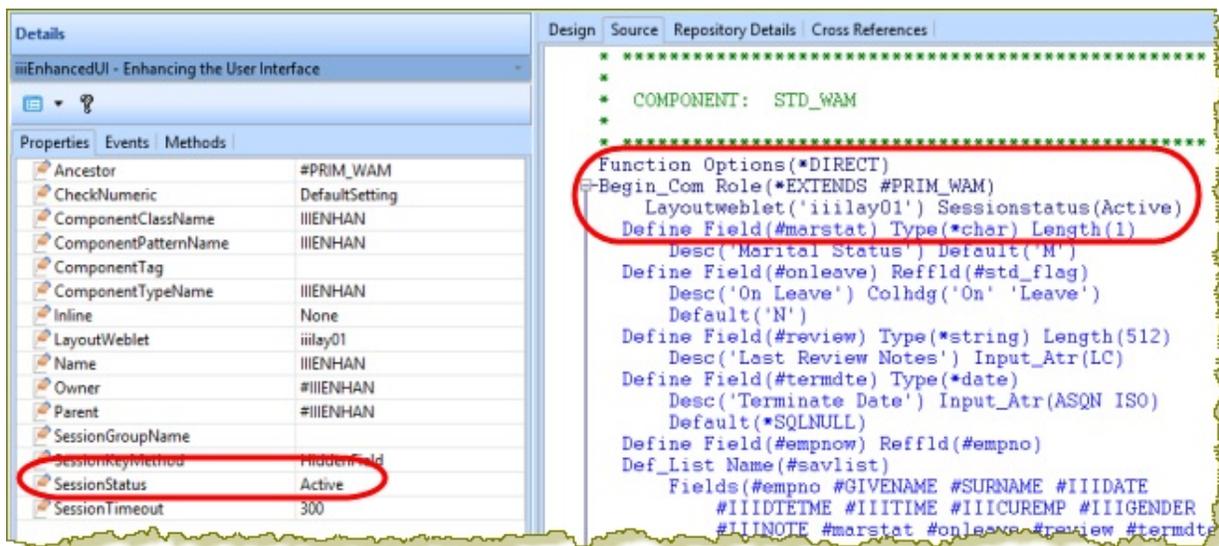
Webroutine Name(showpage) Desc('Enter employee details')  
Endroutine

\* Display the saved working list

Webroutine Name(showlist) Desc('Saved list of employee data')  
Endroutine

3. Note that the **begin** WebRoutine has an **onentry()** keyword of **\*sessionstatus\_none**. This will be the first WebRoutine executed to start session management.

Use the *F7* key to display the WAM properties on the *Details* tab and enable session management:



4. Save your changes.

## Step 4. Define Work Fields and Lists

1. In this step you will define additional fields that will be included on the data entry web page **ShowPage**. The UI for these fields will be manually set up using weblets.

Define the following additional fields at the top of your WAM definition:

```
Define Field(#marstat) Type(*char) Length(1) Desc('Marital Status')
Default('M')
Define Field(#onleave) Reffld(#std_flag) Desc('On Leave') Colhdg('On'
'Leave') Default('N')
Define Field(#review) Type(*string) Length(512) Desc('Last Review Notes')
Input_Atr(LC)
Define Field(#termdte) Type(*date) Desc('Terminate Date') Input_Atr(ASQN
ISO) Default(*SQLNULL)
```

2. Define the following work field and working lists. Ensure that you change **iii** to your initials.

```
Define Field(#empnow) Reffld(#empno)
Def_List Name(#savlist) Fields(#empno #GIVENAME #SURNAME #IIIDAT
Def_List Name(#emplist) Fields((#empno *out) (#GIVENAME *out) (#SURNAME
```

Note that fields in list EMPLIST all have an **\*output** attribute.

3. Define the following global web\_maps:

```
Web_Map For(*both) Fields((#stdrentry *hidden))
Web_Map For(*none) Fields(#savlist) Options(*persist)
```

Note that the list SAVLIST is mapped **\*none**, meaning it is not mapped out to the web page. It has an `Options()` keyword of **\*persist**. With session management enabled, this list will be saved each time the WAM ends, and will be restored before the first WebRoutine is executed. i.e. every time something starts to run in the WAM.

4. Save your changes.

## Step 5. Complete WAM RDMLX

1. The **begin** WebRoutine will be used to start session management and transfer to the **select** WebRoutine. Add the following code to the **begin** WebRoutine:

```
#com_owner.sessionstatus := active  
Message Msgtxt('Session is active')  
Transfer Toroutine(select)
```

2. The **select** WebRoutine display the employee number for input. It also supports push button to clear the saved list. Add the following code to the **select** WebRoutine. Ensure that you change **iii** to your initials.

```
Web_Map For(*output) Fields(#empno)  
Case (#stdrentry)  
When (= C)  
Clr_List Named(#savlist)  
Endcase
```

3. The **showpage** WebRoutine displays all employee fields for input. Employee number is an output field. Work field EMPNOW is mapped as a hidden field to store current employee number.

When initially invoked from the **select** WebRoutine, the employee fields are retrieved.

When invoked from the Save push button, an entry is added to the saved list.

Add the following code to the **showpage** WebRoutine. Change **iii** to your initials.

```
Web_Map For(*both) Fields((#empno *output) #surname #givenname #iiidate #  
Case (#stdrentry)  
* Invoked from select WebRoutine  
When (= S)  
#empnow := #empno  
Fetch Fields(#surname #givenname) From_File(pslmst) With_Key(#empno)  
* Save push button  
When (= U)  
#empno := #empnow  
Add_Entry To_List(#savlist)  
Transfer Toroutine(select)
```

Endcase

4. The **showlist** WebRoutine is invoked via a push button on the select web page. This routine clears the list EMPLIST and populates it from the current saved list SAVLIST. Add the following code to the **showlist** WebRoutine.

```
Web_Map For(*output) Fields(#emplist)
Clr_List Named(#emplist)
Selectlist Named(#savlist)
Add_Entry To_List(#emplist)
Endselect
```

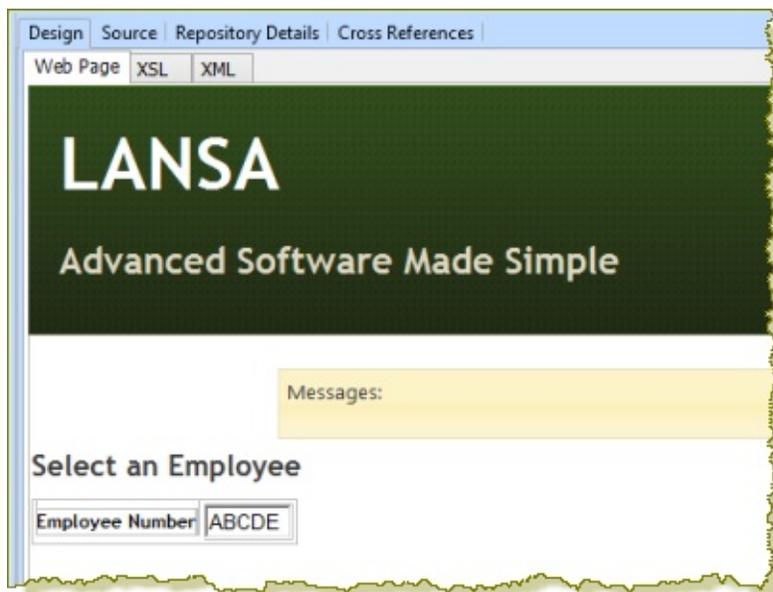
5. Add an event handling routine for invalid session. This will transfer to the **begin** WebRoutine if the WAM is invoked with an invalid or expired session status. The **begin** WebRoutine executes with **\*sessionstatus\_none**. Add the following event routine:

```
Evtoutine Handling(#com_owner.sessionInvalid)
Message Msgtxt('Session has expired')
Transfer Toroutine(begin)
Endroutine
```

6. Compile the WAM.

## Step 6. Design the web pages

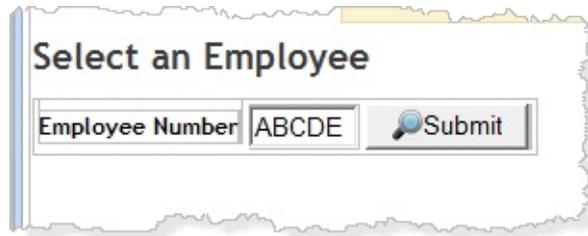
1. Open the **select** WebRoutine in the Design view. It should look like the following:



2. Add a column to the table containing employee number and drop a *Push button with image* weblet into the new cell. Delete the placeholder characters. Set up the push button as follows:

Property	Value
caption	<b>Submit</b>
left_relative_image_path	<b>icons/normal/16/zoom.png</b>
on_click_wname	<b>Showpage</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: S</b>

Your page should look like the following:



- Drop an *jQuery UI AutoComplete* weblet onto the Employee Number field and set up its properties as follows:

Property	Value
sourceWamName	<b>iiiEmpAutoCmpl</b>
sourceWrName	<b>Empno_Prompt</b>
termField	<b>EMPNO</b>
listName	<b>EMP_DD</b>
valueField	<b>EMPNO</b>

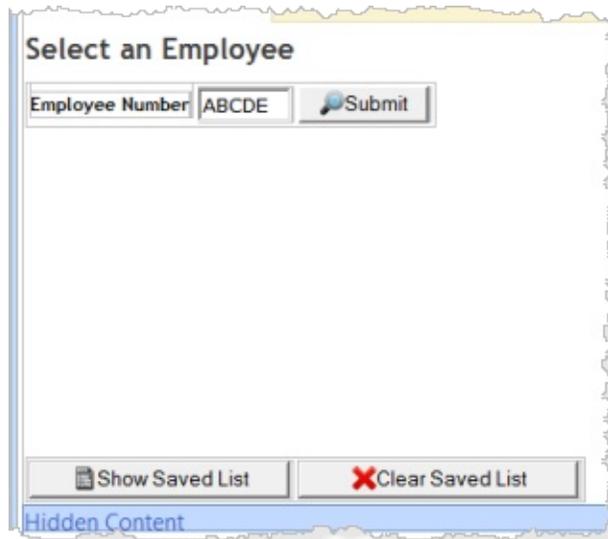
Note that if the WAM **iiiEmpAutoCmpl** is open in the editor, you will be able to select all of these values from a dropdown.

- Use the context menu to Insert HTML / Table with one row and two columns, below the existing table. Use the cursor keys to move to the left of this new table and press *Enter* a number of times to position this table towards the bottom of the central area of the screen.
- Add a push button with images into each cell of the new table. Set up the push buttons as follows:

Property	Value
Caption	<b>Show Saved List</b>
left_relative_image_path	<b>/icons/normal/16/blacklist_16.png</b>
on_click_wrname	<b>Showlist</b>

Property	Value
Caption	<b>Clear Saved List</b>
left_relative_image_path	<b>/icons/normal/16/cross_16.png</b>
on_click_wname	<b>Select</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal Value: C</b>

Your web page should look like the following:



6. Save your changes.
7. Open the **ShowPage** WebRoutine in the design view. Your web page should look like the following:

Design | Source | Repository Details | Cross References

Web Page | XSL | XML

# LANSA

Home | Services

Advanced Software Made Simple

Messages:

### Enter employee details

Employee Number	ABCDE
Employee Surname	aAbBcCdDeEfFgGhHij.
Employee Given Name(s)	aAbBcCdDeEfFgGhHij.
Start Date	01/01/1900
Last Updated	01/01/1900 01:00:00
Comments	aAbBcCdDeEfFgGhHijJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZaAb
Gender	<input type="radio"/> Male <input type="radio"/> Female
Current Employee	<input type="checkbox"/> Caption
Create Time	00:00:00
Marital Status	A
On Leave	A
Last Review Notes	aAbBcCdDeEfFgGhHijJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZaAb
Terminate Date	01/01/1900

- a. Note that Start Date, Last Updated, Comments, Gender, Current Employee and Create Time are all shown as weblents. For example click on the Start Date field and select the Details tab. Note that this is the std\_datepicker weblent.

These weblents are all defined as weblent field visualizations in the *Repository*.

- b. Select **Last Review Notes** and then **Terminate Date** and check their definition using the Details tab. Notice that they have also been defined in the web page as weblents (std\_varchar and std\_DatePicker respectively) based on their field type. These are fields you defined in the WAM.
- c. The single character (Alphanumeric) fields **Marital Status** and **On Leave** are input fields. In the *Design* view you will replace these with weblents.

8. Select the field **Start Date** and set up its properties as follows:

Property	Value
changeMonth	<b>True</b>
changeYear	<b>True</b>
showOtherMonths	<b>True</b>
selectOtherMonths	<b>True</b>

You will be able to see the effect of these settings once you test the WAM.

Change the *dateFormat* property to be correct for your region. **mm/dd/yyyy** is the default value.

Also change the *dateFormat* for fields **Terminate Date** and **Last Updated** to suit your region.

9. Select the **Comments** field and set up its properties as follows:

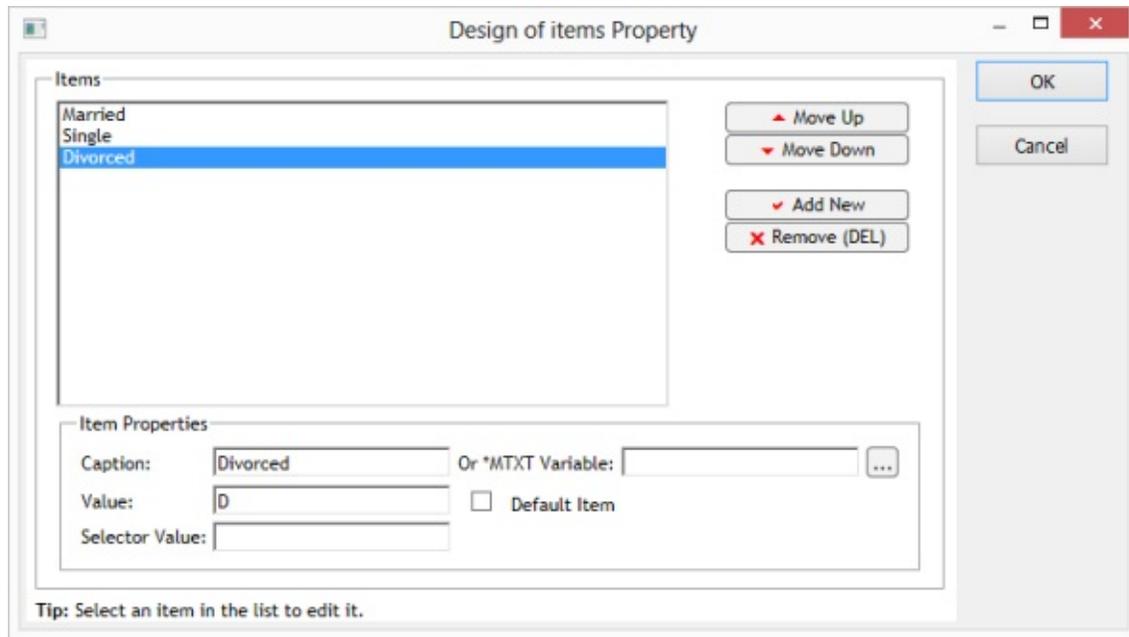
Property	Value
Type	<b>memo</b>
word_wrap	<b>soft</b>

Note that the **Comments** field is now a memo box.

10. Select the **Current Employee** checkbox and change its caption to ''. The caption should now be blank.

Note that the *oncode* and *offcode* properties for the check box weblet are values of **Y** and **N** which are correct for this field. These values could be changed if necessary.

11. Drop a *Radio Group* weblet onto the field **Marital Status**. On the *Details* tab, in the *items* property's value column, use the *Ellipsis* button to open the *Design of...* dialog to define three captions and values as shown:



Caption	Value
Married	M
Single	S
Divorced	D

12. Drop a *Check Box* weblet onto the **On Leave** field and set its caption to ''.

Once again the default *oncode* and *offcode* values are suitable for this field.

13. Select the **Last Review Notes** field (std\_varchar weblet) and set its properties as for the Comments field in sub step 9. [Select the Comments field and set up its properties as follows:](#)

14. Select the **Terminate Date** field (std\_DatePicker weblet) and set its properties as for the Start Date in sub step 8. [Select the field Start Date and set up its properties as follows:](#)

15. Save your changes.

16. Drop a *Push Button with Image* weblet into the bottom right of the table, next to the Terminate Date field. Set up its properties as follows:

Property	Value

Caption	<b>Save</b>
left_relative_image	<b>icons/normal/16/check_mark_16.png</b>
on_click_wname	<b>Showpage</b>
submitExtraFields	<b>Field Name: STDRETRY</b>
	<b>Literal Value: U</b>

17. In this step you will add a link to the **Show Saved List** page to return to the Select Employee Number page.

- a. Open the **showlist** WebRoutine in the Design view.
- b. Drop a *Push Button with Images* onto the page, below the list.
- c. Set up the push button properties as follows:

<b>Property</b>	<b>Value</b>
caption	<b>Select Employee</b>
left_relative_image	<b>icons/normal/16/zoom.png</b>
on_click_wname	<b>Select</b>
submitExtraFields	<b>Field Name: STDRETRY</b>
	<b>Literal Value: T</b>

**Note:** The *dateFormat = Auto* property for the DatePicker weblet, will be correct for most applications. The date format will be determined by the partition's ISO Language setting. For the UK this should be **en-GB** since **en** will give DD/MM/YYYY.

18. Save your changes.

## Step 7. Test the WAM

The WAM should be executed by calling the begin WebRoutine. However, note that calling any WebRoutine initially, will transfer control to the begin WebRoutine, via the session invalid event handling routine.

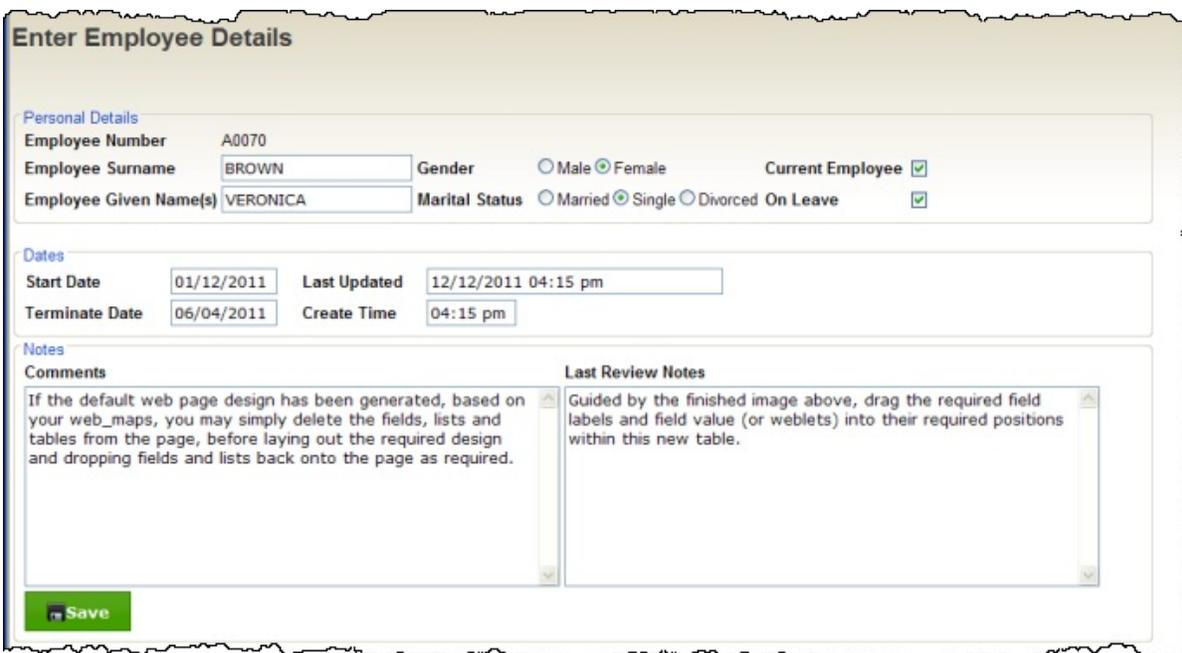
- On the *Select an Employee* page, you can enter a valid employee number by typing a character and selecting from the response in the AutoComplete weblet. Note that all employee numbers begin with 'A' followed by a four digit number. The weblet is set up to invoke the response WebRoutine with a delay of 150ms after you stop typing and after 2 characters have been entered.
- Data can then be entered on the Enter Employee Data web page. Note that initially this page will contain employee number, surname and given name only. You can enter other data via the weblets that support these field values.
- Note that the Start Date DatePicker allows year and month to be selected, within a 10 year range of current date. These limits can be adjusted by changing the weblet properties.
- The Comments std\_varchar weblet applies word wrap as you type beyond the width of the control.
- Check box and Radio Group return values according to your selection.
- The Save button will return to the Select an Employee web page.
- Enter data for more than one employee.
- On the *Select an Employee* page, use the *Show Saved List* button to display the current saved list of entries, via the **showlist** WebRoutine.
- On the *Select an Employee* page, use the *Clear Saved List* to clear the saved list and then repeat your testing

## Step 8. Improve the ShowPage Page Design

In this step you will change the layout of the *Enter Employee Details* page, by moving the fields into logical groups on the page. You will also insert `<fieldset>` tags around each group of fields.

- The fieldset tag displays a "group box" around an area of the screen. The fieldset tag may include a `<legend>` tag that defines a caption displayed at the top left corner of the box.
- The fieldset tag must enclose the complete table, if a table is being used to lay out a number of fields and labels.
- The fieldset tags may not enclose a number of rows within a table.
- For more information on all HTML and related topics, see [www.w3schools.com](http://www.w3schools.com)

The finished page design (showpage) will look like the following:



The screenshot shows a web form titled "Enter Employee Details". It is organized into three main sections, each enclosed in a fieldset:

- Personal Details:** Contains fields for Employee Number (A0070), Employee Surname (BROWN), Employee Given Name(s) (VERONICA), Gender (radio buttons for Male and Female, with Female selected), Marital Status (radio buttons for Married, Single, and Divorced, with Single selected), Current Employee (checkbox checked), and On Leave (checkbox checked).
- Dates:** Contains fields for Start Date (01/12/2011), Terminate Date (06/04/2011), Last Updated (12/12/2011 04:15 pm), and Create Time (04:15 pm).
- Notes:** Contains two text areas: "Comments" and "Last Review Notes". The "Comments" area contains text about default web page design. The "Last Review Notes" area contains text about guided design.

A green "Save" button is located at the bottom left of the form.

Figure 1. Enter Employee Details

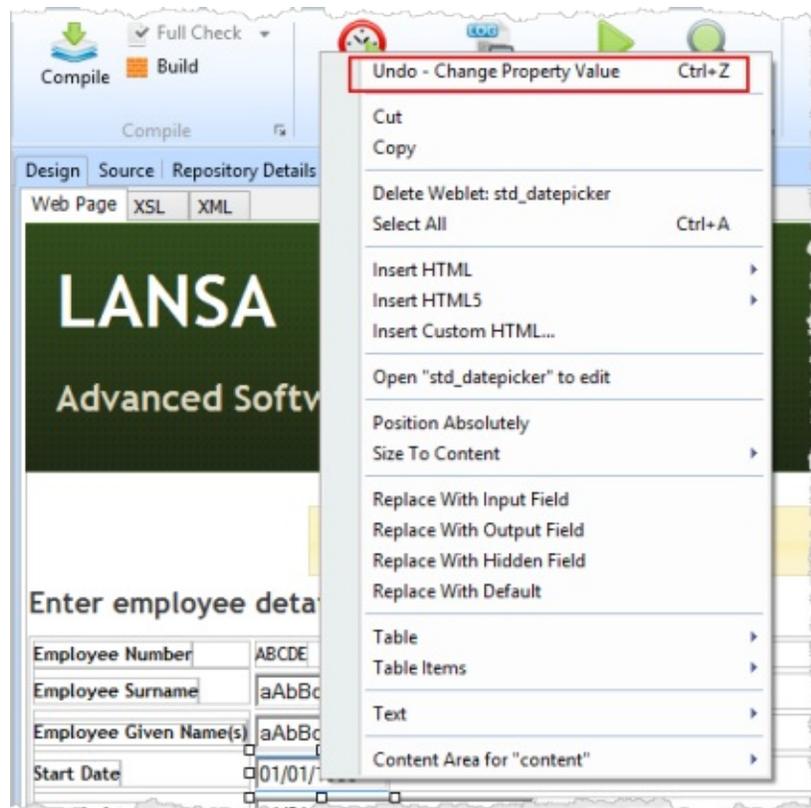
To redesign this web page, follow the details steps provided. However, bear in mind that other approaches could have been followed:

- If the WAM is initially compiled **before** any web\_maps are defined, the web page will be blank.

- Recompiling the WAM when all the required web\_maps have been defined, will list the field and list mappings on the *WebRoutine Output* tab. The page may then be defined in the *Design* view, by dragging and dropping fields into their required locations.
- If the default web page design has been generated, based on your web\_maps, you may simply delete the fields, lists and tables from the page, before laying out the required design and dropping fields and lists back onto the page as required.
- In this exercise, you will insert new elements into the page (tables) and drag and drop fields and labels on the page in the *Design* view to achieve the result required.
- As you work through this part of the exercise, refer Figure 1 for guidance on the end result required.

## Editor Undo / Redo Feature

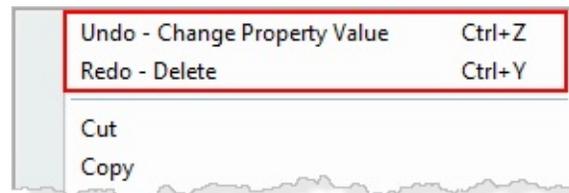
The editor which enables you to design your web page is actually editing the underlying XSL code which will transform XML into HTML at runtime. The Undo/Redo feature has been enhanced in V13 SP1 to provide more powerful and easy to use functionality.



The Start Date was changed, making the *dateFormat* = **dd/mm/yyyy**. The context menu shows that *Undo* will remove the property change. Alternatively, Ctrl+Z could be used.



Another option is to use the *Undo* button in the *Clipboard* group on the *Home* ribbon.



Details of a number of changes are stored. The above example shows that an *Undo* was performed after a field label had been deleted. *Redo* will delete this label text again. The Start Date property change could also be removed.

1. Open the **showpage** WebRoutine in the *Design* view.
2. In this step you will create the *Personal Details* area containing employee number, surname, given name, gender, marital status, current employee and on leave fields.

Insert a table at the bottom of the page with 3 rows and 6 columns

3. Guided by the finished image above, drag the required field labels and field value (or weblets) into their required positions within this new table.

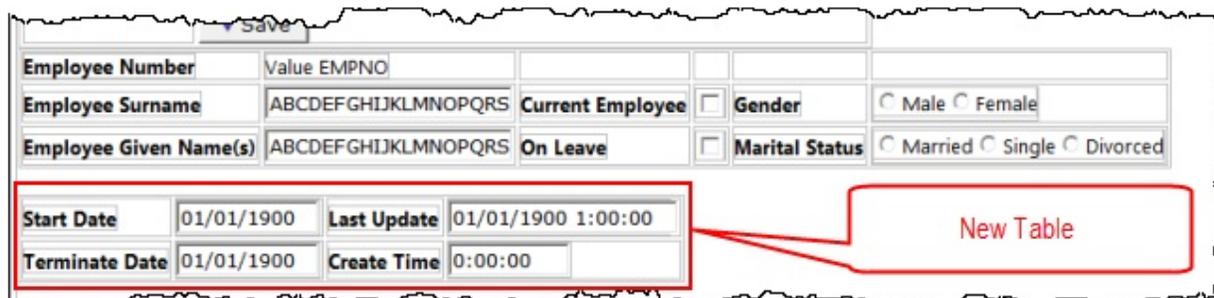
**Note:** You may find an easier approach is to *Cut* (Ctrl+X) each element from the existing table and then *Paste* (CTRL+V) it into the new table.

4. In the new table, select each table cell (that is, the <td> tag) containing a *field label* and set its *class* property to **caption**.
5. Remove the place holder characters where necessary.

Your page should now look like the following:



6. Save your changes.
7. Click in the top row of the original table, which is now empty, and use the context menu to use *Table Items / Delete Rows...* to delete **3** rows. Repeat this step as necessary in the other empty rows, where you have dragged fields into the new table.
8. Insert another new table at the bottom of the page, with 2 rows and 4 columns. Drag the date and time fields and labels into the new table, or *Cut* and *Paste* each item.
9. Select each cell containing a label and change its *class* to **caption**.
10. Remove the placeholder characters. Your page should now look like the following:

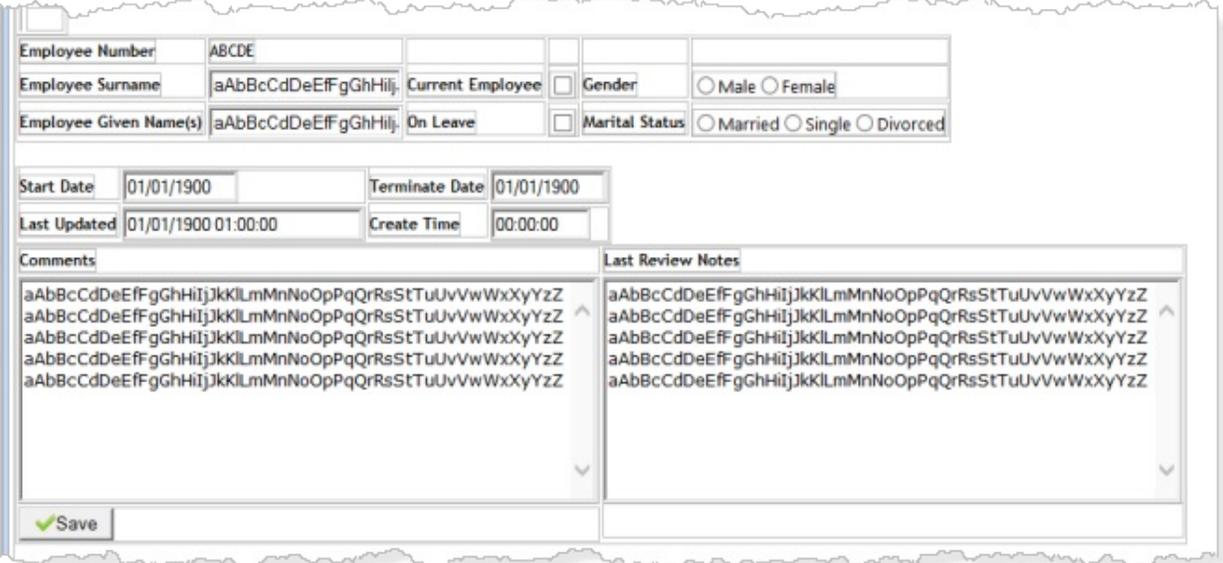


11. Once again make the original table smaller by deleting the empty rows. This step is optional, since you will eventually delete this whole table. Deleting rows makes it easier to drag items to the new table.
12. Insert a new table at the bottom of the page with 3 rows and 2 columns. Drag the comment and notes fields and labels into this new table. Review the image in 16. before you complete this step.
13. Select each cell containing a label and change its *class* to **caption**. Remove the place holder characters.

14. Add some \*\* characters to the original table, so that it remains easily visible and then drag the Save button into the bottom left hand cell.

15. Drag the Save button into the bottom left cell of the new table.

16. Save your changes. Your page should look like the following:



17. Select the corner of the original table to select the whole table, and delete it.

18. Save your changes.

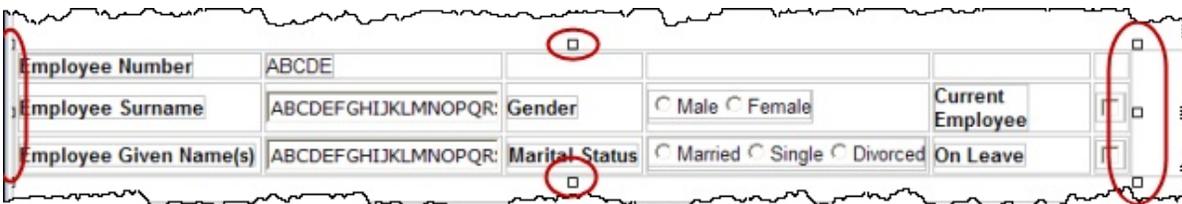
## Step 9. Insert a fieldset around each table

In this step you will insert a fieldset around each table. This will display a round cornered group box around each area. You will also add a legend tag in each fieldset to display a caption.

Lastly, in the *Design* view you will insert a div around each fieldset and size it. You are doing this to size the fieldset that will otherwise have the same width as its container. It is possible to size the fieldset by giving it a class and using CSS, but a bug in IE would then display the fieldset with square corners. Adding the div avoids this issue.

1. Select the top table containing employee details. Select a corner of the table. Alternatively, select anywhere in the table and use the *Outline* tab, to select the table itself.

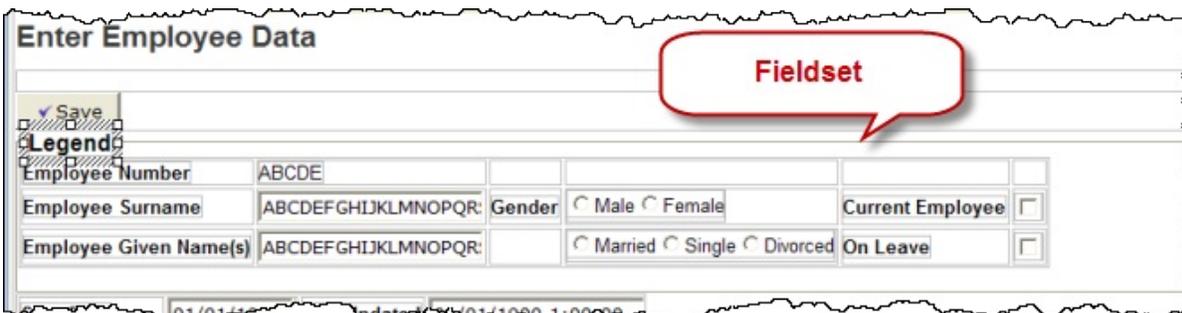
The Design view should now look like the following:



The image shows a web form in Design view. The form contains a table with the following fields: Employee Number (text), Employee Surname (text), Employee Given Name(s) (text), Gender (radio buttons for Male and Female), Marital Status (radio buttons for Married, Single, and Divorced), Current Employee (checkbox), and On Leave (checkbox). The table is surrounded by a jagged, torn-paper-like border. Small square handles are visible at the corners of the table, and a red circle highlights one of these handles.

Note the "handles" around the table that is currently selected.

2. Use the right mouse button over one of these "handles" and select *Insert HTML / Fieldset* to insert a fieldset around this table. Your design should look like the following:



The image shows the same web form as in the previous screenshot, but now a red rounded rectangle labeled "Fieldset" is drawn around the table. The form also includes a "Save" button and a "Legend" text area. The jagged border remains around the entire form area.

3. Click in the Legend text area and replace it with **Personal Details.**
4. Select the table containing Start Date and insert a fieldset around this table, with and replace Legend text with **Dates.**
5. Select the table containing Comments and insert a fieldset around this table

with the legend **Comments**.

6. Save your changes.

Details					
Employee Number	Value EMPNO				
Employee Surname	ABCDEFGHIJKLMNOPS	Current Employee	<input type="checkbox"/>	Gender	<input type="radio"/> Male <input type="radio"/> Female
Employee Given Name(s)	ABCDEFGHIJKLMNOPS	On Leave	<input type="checkbox"/>	Marital Status	<input type="radio"/> Married <input type="radio"/> Single <input type="radio"/> Divorced

7. Run your application to see the results in the browser. Enter some data. Your page should look like the following:

**LANSA**  
Advanced Software Made Simple

Home Services Contact About

### Enter employee details

**Personal Details**

Employee Number: A0070

Employee Surname: BROWN Current Employee:  Gender:  Male  Female

Employee Given Name(s): VERONICA On Leave:  Marital Status:  Married  Single  Divorced

**Dates**

Start Date: 03/04/2013 Terminate Date: 30/04/2013

Last Updated: 12/04/2013 05:44 pm Create Time: 05:44 pm

**Comments**

**Comments**

consequat. Ut magna pulvinar ac at nisl. Mauris augue eu. Odio commodo lacinia habitasse nec urna imperdiet pede sapien in amet vitae porttitor adipiscing arcu sit condimentum sed sed ultrices vestibulum. Aliquam elit sequi molestie nibh per. Eget hac amet libero lorem eget. Amet nec nisl. Amet ut a. Turpis suspendisse enim. Pede tempora ipsum nam a nihil sit ut massa vel arcu suspendisse. Nunc eget orci. Non quia lacus. Consequat mollis consectetur. Tempor dignissim ultricies. Felis interdum dapibus lacus accumsan fermentum.

**Last Review Notes**

Orci augue vel. Quis eum lectus. Consectetur lacus enim quisque ac tincidunt fringilla semper nonummy. Neque dui lacinia. Aliquet nec eros ut non per. Faucibus ut adipiscing aliquam porttitor purus a itaque eget. Mauris vel pede sem in quo. Purus at leo posuere venenatis donec erat dis mauris nullam ut vestibulum.

Save

Notice that the group boxes span the full width of the web page. The screen shot has minimized the page width. With the browser maximised on a large screen the blank areas on the right look a little unfinished.

The web site <http://lorem-ipsam.perbang.dk/> may be used to generate 'mock latin' text to fill up your memo boxes.

8. With the **showpage** WebRoutine open in the *Design* view, select each fieldset border and use the context menu to *Insert* a **Div**. Check you have selected the fieldset by looking at the *Properties* sheet on the *Details* tab. It should display **<fieldset>** at the top. Use the context menu to *Insert HTML / DIV* around the fieldset.
9. With the Div selected use the *Details* tab, to expand the *Style* properties and change the *width* to **80%**.
10. Repeat the last two steps for the other two fieldsets.
11. Save your changes.
12. Run the application. Your web page should now look like the following:

**Enter Employee Data**

Details

Employee Number: A0070

Employee Surname: BROWN    Current Employee:     Gender:  Male  Female

Employee Given Name(s): VERONICA    On Leave:     Marital Status:  Married  Single  Divorced

Dates

Start Date: 31/01/2001    Last Update: 25/08/2011 10:53 am

Terminate Date: 10/08/2006    Create Time: 10:53 am

Notes

**Comments**

In euismod signiferumque nam, graecis assueverit id nec. Viderer dolores has ne, ad eam solum choro. Usu at nibh praesent. Mollis tractatos te vix, ex eos tota aperiam. Ne partem indoctum his, his perfecto iudicabit an. Reque utamur facilis his an. Id omnes soluta dissentiet pro.

**Last Review Notes**

Lorem ipsum dolor sit amet, quis persius insolens id sed. Simul zril intellegat in duo. Enim blandit per ei, et rebum decore mea, an vix sale munere. Quem saperet honestatis at his. Falli debet necessitatibus sit et, cum wisi elltr no. Occurreret prodesset est id, mea ea oratio audire.

## Summary

### Important Information

- Only weblet field visualizations that are likely to change **very infrequently**, should be defined in the Repository
- Field types Date, DateTime, Time and String will be given weblet visualizations by default.
- The *Design* view provides a powerful graphical editor, that enables the developer to rapidly refine the page design and layout.
- You will always require some knowledge of HTML and style sheets (CSS) to be able to work with web page design.
- Excellent tutorials for all web languages are available at [www.w3schools.com](http://www.w3schools.com)
- If your application is aimed at the consumer, you will probably keep the page design much simpler than this example, possibly spread over two or three pages.

### Tips & Techniques

- This exercise has demonstrated some aspects of some of the weblets available. The Web Application Modules guide provides more detailed information for all of the weblets.
- You should be aware that fields supported by weblets should always be defined with a suitable default value. The weblet only returns a value when the user selects a value. e.g. only when the user checks or un-checks a check box, does the weblet return a value.

### What You Should Know

- How to make use of the available weblets to enhance the user interface.

## **WAM090 - Using a List Row Weblet**

### **Objectives**

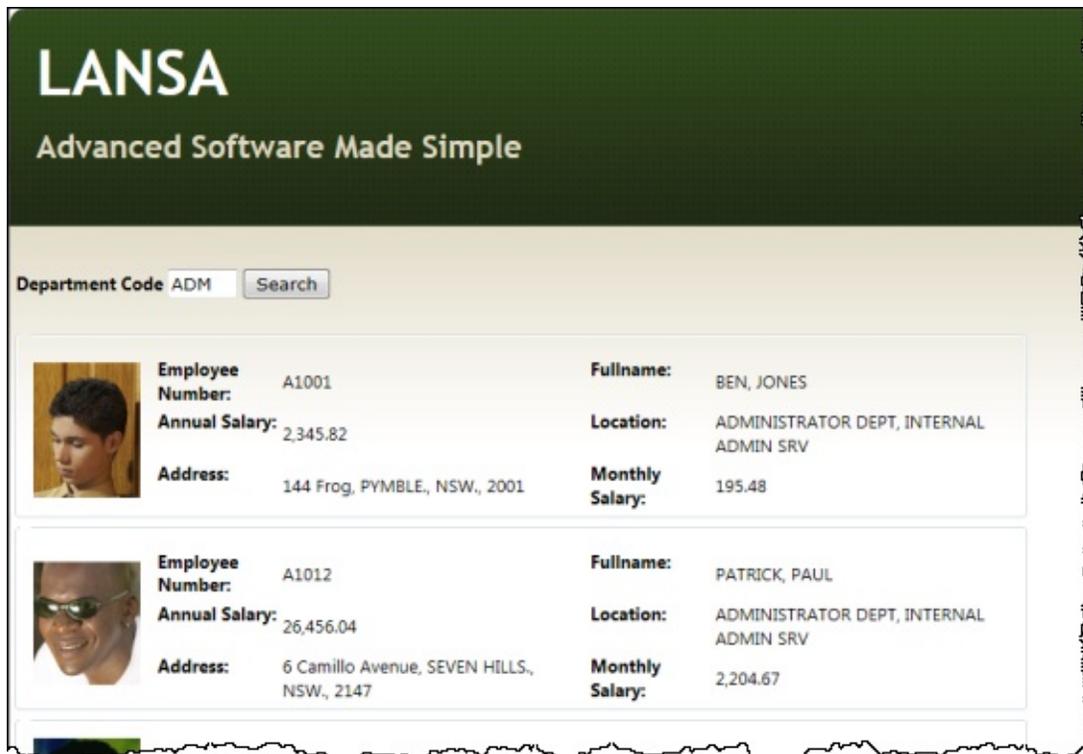
In the exercises which you have completed up to now, lists have always been formatted as a simple table with column headings and data shown in columns. This exercise demonstrates how you can present a list as a series of formatted rows. Your list output can in fact be formatted almost any way you wish. Achieving the result you want, will depend on your skill with HTML and XSL transformation.

This exercise provides a custom weblet (`iii_ListRow`) for you to use. Although you will not be examining its XSL and HTML, it will demonstrate how list data can be treated in a different way.

### **The List Row Weblet (`iii_ListRow`)**

The main concept to keep in mind in this exercise is that list output consists of a repeating set of data. The row weblet `iii_ListRow` allows you to associate a working list with the weblet and then assign the fields within this list to parameters for each row. The working list needs to contain:

- The data fields – the values you want to output
- Fields holding the labels (or captions) for each of the output fields.
- A field containing a path and file name for a small employee image.
- Each row is wrapped in a `<fieldset>` tag. The list also contains a field and label field for the `<legend>` tag that provides the caption at the top left corner of the group box that the fieldset provides.
- The WebRoutine should map the working list with a `*private` attribute, since the List Row weblet will format the output.



The WAM will be a simple enquiry that lists employees by department.

To complete this exercise, complete the following:

Step 1. Create the List Row Weblet – [iii\\_ListRow](#)

Step 2. Create WAM [iiiUseListRowWeblet](#)

Step 3. Set Up the Web Page

[Summary](#)

## Before You Begin

Complete the preceding exercises in this workshop.

Step 3 of this WAM requires a set of small images for employees. If you haven't already done so, you can download these from the documentation page of the LANSA web site as described in the *Before You Begin* section at the start of these [WAM Tutorials](#).

Copy the images from the PHOTOS folder into to your web server /images path, such as: C:\Program Files (x86)\LANSA\WebServer\images. If you are running your WAMs on the IBM i, copy this folder to:

/LANSA\_<PGMLIB>/webserver/images where <PGMLIB> is your LANSAs program library.

## Step 1. Create the List Row Weblet – iii\_ListRow

1. Create a new weblet as shown:



The screenshot shows a 'New Weblet' dialog box with the following fields and values:

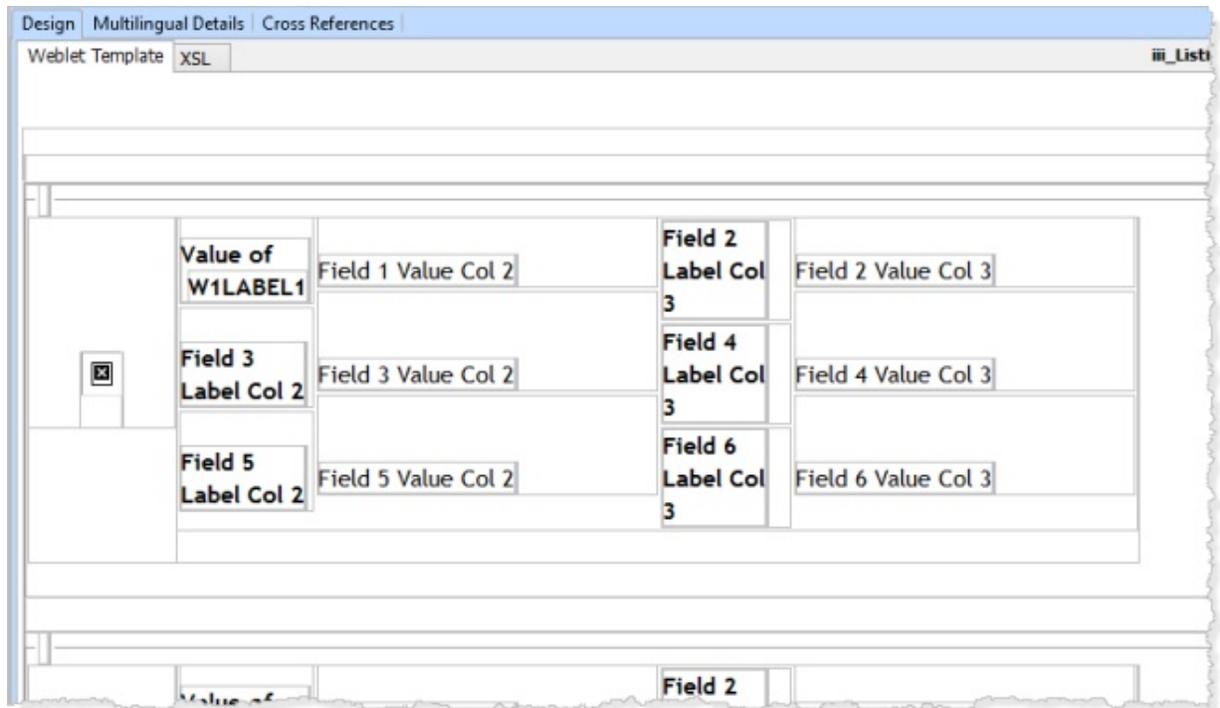
Field	Value
Name	iii_ListRow
Description	Training List Row Weblet
Weblet Group	Custom Weblets
Layout Weblet	<input type="checkbox"/>

Buttons: Create, Cancel

Give the weblet a *Weblet Group* of **Custom Weblets**.

If the group *Custom Weblets* does not already exist, type the name into the Weblet Group combo box and a new group will be created.

2. Copy the weblet XSL code from [WAM090. Appendix](#), at the end of this exercise and **replace** the default code in your new weblet.
3. Save the new weblet definition.
4. Select the *XSL* tab and use the *Replace* dialog to replace all occurrences of **iii\_ListRow** with the same name using your initials in place of **iii**. There should be 5 occurrences.
5. Save your changes.
6. Select the *Design* view. Your weblet design should look like the following:



A row is defined with an image on the left hand side and data fields and label fields arranged in three rows with four columns.

## Step 2. Create WAM **iiiUseListRowWeblet**

1. Create a new WAM:

Name: **iiiUseListRowWeblet**

Description: **Using a List Row Weblet**

Layout Weblet: **iiilay01**

2. Add the following definitions after the Begin\_Com:

```
Define Field(#fulladdr) Reffld(#std_textl)
Define Field(#location) Reffld(#std_textl)
Define Field(#DESC1) Reffld(#std_desc) Default("Name: ")
Define Field(#DESC2) Reffld(#std_desc) Default("Location: ")
Define Field(#DESC3) Reffld(#std_desc) Default("Address: ")
Define Field(#DESC4) Reffld(#std_desc) Default("Start Date: ")
Define Field(#DESC5) Reffld(#std_desc) Default("Monthly Salary: ")
Define Field(#DESC6) Reffld(#std_desc) Default("Annual Salary: ")
Define Field(#DESC7) Reffld(#std_desc) Default("Employee Number: ")
*
Group_By Name(#empdata) Fields(#empno #SURNAME #GIVENAME #AD
Def_List Name(#emplist) Fields(#empno #fullname #fulladdr #location #SAL.
*
Web_Map For(*both) Fields((#stdrentry *hidden))
WebRoutine Name(list)
Web_Map For(*both) Fields(#deptment) /* comment */
Web_Map For(*output) Fields((#emplist *private))
Case (#stdrentry)
When (= S) /* Build list of employees */
Clr_List Named(#emplist)
Select Fields(#empdata) From_File(pslmst1) With_Key(#deptment) Nbr_Keys
Fetch Fields(#deptdesc) From_File(deptab) With_Key(#deptment) Keep_Last(
Fetch Fields(#secdesc) From_File(sectab) With_Key(#deptment #section) Kee
#std_textl := 'PHOTOS/' + #EMPNO + '.jpg'
#fullname := #givename + ', ' + #surname
#fulladdr := #address1 + ', ' + #address2 + ', ' + #address3 + ', ' + #postcode.ass
#location := #deptdesc + ', ' + #secdesc
Add_Entry To_List(#emplist)
Endselect
Otherwise
```

Message Msgtxt('Enter a department code and Search')

Endcase

Endroutine

3. Review the supplied code.

The only unusual part of this WAM is that it defines a number of work fields used to define the labels required for the fields to be output in each row.

The working list EMPLIST contains the required fields for each employee, selected from the logical file PSLMST1 with a key of DEPARTMENT.

4. Compile the WAM.

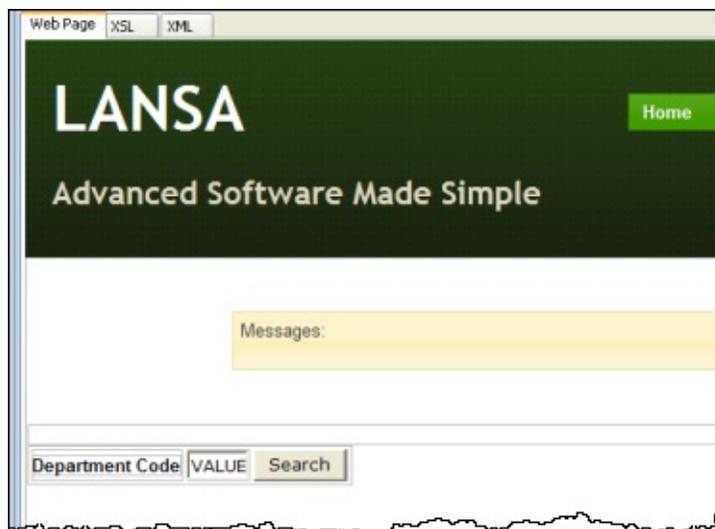
### Step 3. Set Up the Web Page

1. Open the **list** WebRoutine in the *Design* view.
2. Add a column to the table containing the department code.
3. Drop a Push button into the new cell and set up its properties:

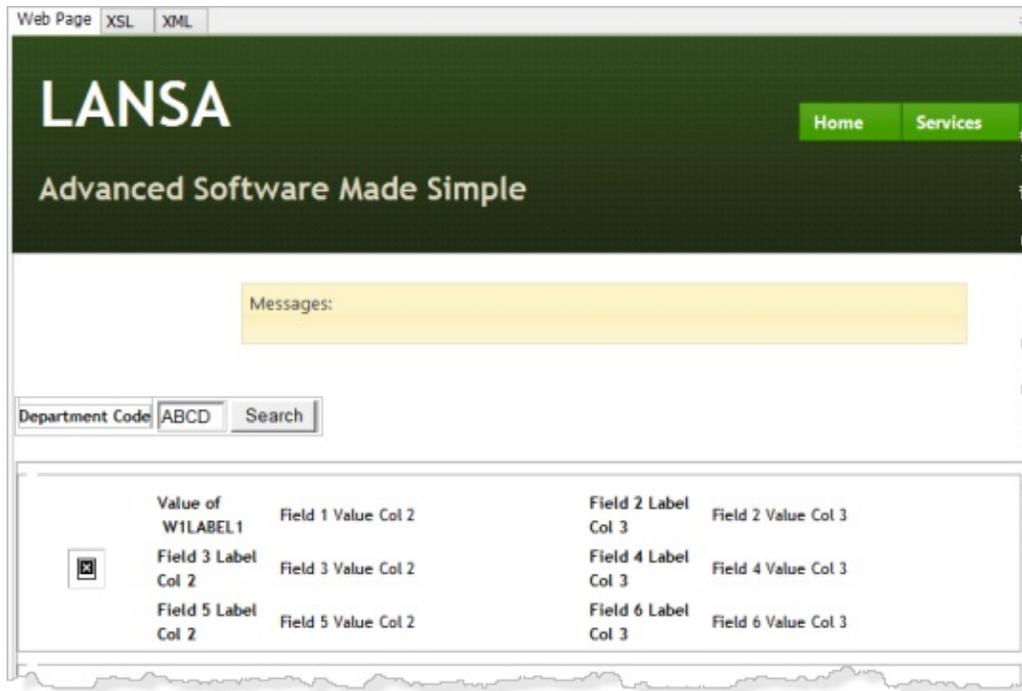
Property	Value
Caption	<b>Search</b>
On_click_wrname	<b>list</b>
submitExtraFields	<b>Field name: STDREENTRY</b>
	<b>Literal Value: S</b>

4. Save your changes.

Your page should look like the following:



5. On the *Favorites / Weblet Templates* tab, select *Custom Weblets* in the top combo box. Drop the *Training List Row Weblet* onto the page. Your page should look like the following:



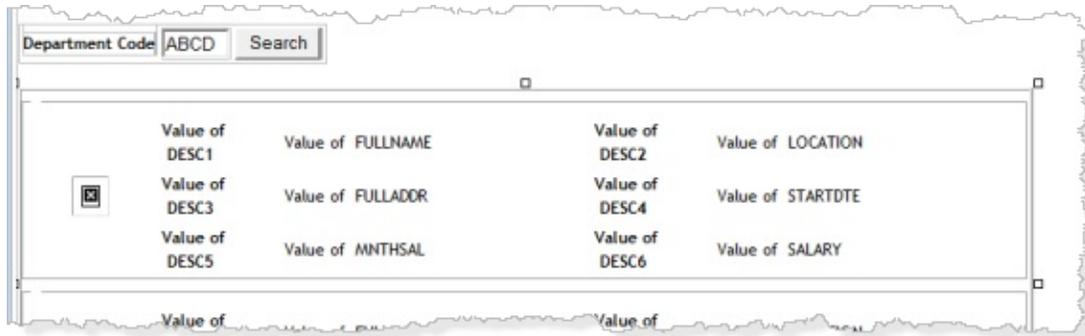
6. Select the List Row Weblet, Select the *Details* tab and set up its properties as follows:

Property	Value
Listname	<b>EMPLIST</b>
List_field_label_1	<b>DESC1</b>
List_field_value_1	<b>FULLNAME</b>
List_field_label_2	<b>DESC2</b>
List_field_value_2	<b>LOCATION</b>
List_field_label_3	<b>DESC3</b>
List_field_value_3	<b>FULLADDR</b>
List_field_label_4	<b>DESC4</b>
List_field_value_4	<b>STARTDTE</b>
List_field_label_5	<b>DESC5</b>
List_field_value_5	<b>MNTHSAL</b>
List_field_label_6	<b>DESC6</b>

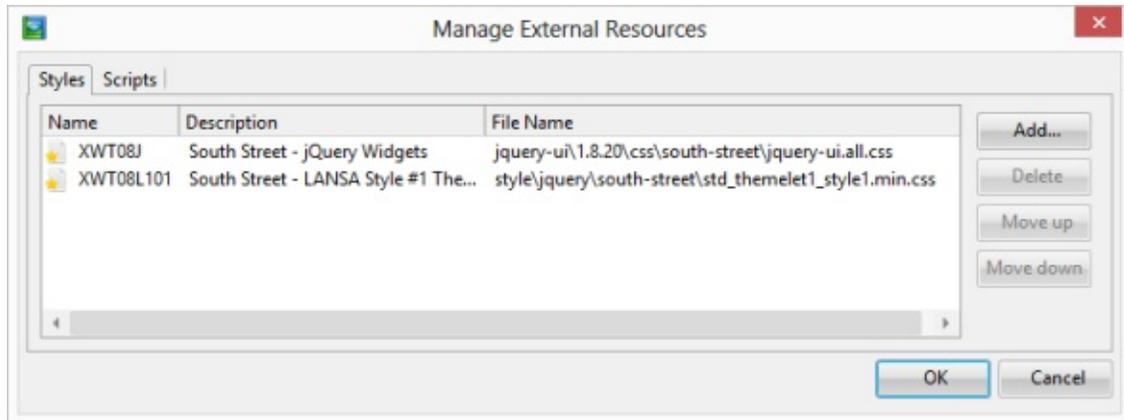
List_field_value_6	<b>SALARY</b>
List_field_label_7	<b>DESC7</b>
List_field_value_7	<b>EMPNO</b>
List_image_field	<b>STD_TEXTL</b>

7. Save your changes.

Your design should look like the following:



- This step will give the list row weblet the correct *External Resources* to match your selected page layout theme. See *WAM060 – Employee Maintenance*, [Step 4. Define the Details WebRoutine](#) where you used *Cross References* for your layout weblet to find the *External Resources* used by your chosen theme.
- With the *Design* view open, select the *Design ribbon / External Resources* button. Knowing your external resources (the cascading style sheets) used by your theme, delete the default files used by the weblet code provided and add the files for your theme.



10. Save your changes.
11. This step requires a set of small images for employees. If you haven't already done so, you can download these now following the instructions in *Before You Begin* at the beginning of this exercise.
12. Test your WAM and list employees for department code ADM. At the moment your list will be displayed on the left hand side of the page.
13. In the *Design* view select the List Row weblet and use the context menu to *Insert / Div*. With the div selected, change its *align* property to **center**. Save your changes.
14. Test your WAM. The list will now be displayed in the center of the web page.

## Summary

### Important Observations

- Lists may be presented in any format by making use of the ability to create your own weblets. This will require good HTML knowledge and a working knowledge of XSLT.
- With some modifications the example given here could be used as part of a shopping application.
- Later exercises will cover creating your own simple weblet.
- Once you have completed *WAM105 – Create Your Own Weblet* and *WAM110 – Create Your Own Layout*, you should review the List Row weblet provided for this exercise. Although it is reasonably complex, once you understand the structure of weblets, you should be able to follow its logic by examining its different sections.

### Tips & Techniques

- Lists can also be formatting as a single row. This is useful way to present a variable number of options

### What You Should Know

- Custom weblets provide a highly flexible way to enhance the user interface of your web applications. Although the investment in creating and testing a custom weblet may be quite high, as a reusable component, it will pay off many times and simplify work for other developers.

## WAM090. Appendix

### List Row Weblet iii\_ListRow (XSL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- LANSA for the Web -->
<xsl:transform version="1.0" exclude-result-prefixes="lxml wd"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
xmlns="http://www.w3.org/1999/xhtml">
<xsl:import href="std_types.xsl" />
<xsl:import href="std_keys.xsl" />
<xsl:import href="std_anchor.xsl" />
<xsl:output method="xml" omit-xml-declaration="yes" encoding="UTF-8"
indent="no" />
<wd:external-resources>
<wd:style name="XWT08J" />
<wd:style name="XWT08L101" />
</wd:external-resources>
<wd:definition>
<wd:group name="Custom Weblets" />
</wd:definition>
<lxml:data>
<lxml:list name="">
<lxml:list-header>
<lxml:header name="W1LABEL1">
<lxml:heading-1>List Field Label 1</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1FIELD1">
<lxml:heading-1>List Field 1</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1LABEL2">
<lxml:heading-1>List Field Label 2</lxml:heading-1>
<lxml:heading-2 />
```

```
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1FIELD2">
<lxml:heading-1>List Field 2</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1LABEL3">
<lxml:heading-1>List Field Label 3</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1FIELD3">
<lxml:heading-1>List Field 3</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1LABEL4">
<lxml:heading-1>List Field Label 4</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
<lxml:header name="W1FIELD4">
<lxml:heading-1>List Field 4</lxml:heading-1>
<lxml:heading-2 />
<lxml:heading-3 />
</lxml:header>
</lxml:list-header>
<lxml:list-entries>
<lxml:entry>
<lxml:column name="W1LABEL1"
id="SAMPLE_LIST.0001.W1LABEL1">Field 1 Label Col 2</lxml:column>
<lxml:column name="W1FIELD1"
id="SAMPLE_LIST.0001.W1FIELD1">Field 1 Value Col 2</lxml:column>
<lxml:column name="W1LABEL2"
id="SAMPLE_LIST.0001.W1LABEL2">Field 2 Label Col 3</lxml:column>
<lxml:column name="W1FIELD2"
id="SAMPLE_LIST.0001.W1FIELD2">Field 2 Value Col 3</lxml:column>
<lxml:column name="W1LABEL3"
```

id="SAMPLE\_LIST.0001.W1LABEL3">Field 3 Label Col 2</lxml:column>  
<lxml:column name="W1FIELD3"  
id="SAMPLE\_LIST.0001.W1FIELD3">Field 3 Value Col 2</lxml:column>  
<lxml:column name="W1LABEL4"  
id="SAMPLE\_LIST.0001.W1LABEL4">Field 4 Label Col 3</lxml:column>  
<lxml:column name="W1FIELD4"  
id="SAMPLE\_LIST.0001.W1FIELD4">Field 4 Value Col 3</lxml:column>  
<lxml:column name="W1LABEL5"  
id="SAMPLE\_LIST.0001.W1LABEL3">Field 5 Label Col 2</lxml:column>  
<lxml:column name="W1FIELD5"  
id="SAMPLE\_LIST.0001.W1FIELD3">Field 5 Value Col 2</lxml:column>  
<lxml:column name="W1LABEL6"  
id="SAMPLE\_LIST.0001.W1LABEL4">Field 6 Label Col 3</lxml:column>  
<lxml:column name="W1FIELD6"  
id="SAMPLE\_LIST.0001.W1FIELD4">Field 6 Value Col 3</lxml:column>  
<lxml:column name="W1IMAGE"  
id="SAMPLE\_LIST.0001.W1IMAGE">IMAGE\_FILE</lxml:column>  
<lxml:column name="W1LABEL7"  
id="SAMPLE\_LIST.0002.W1LABEL7">Field 7 Label  
Legend</lxml:column>  
<lxml:column name="W1FIELD7"  
id="SAMPLE\_LIST.0002.W1FIELD7">Field 7 Value  
Legend</lxml:column>  
</lxml:entry>  
<lxml:entry>  
<lxml:column name="W1LABEL1"  
id="SAMPLE\_LIST.0001.W1LABEL1">Field 1 Label Col 2</lxml:column>  
<lxml:column name="W1FIELD1"  
id="SAMPLE\_LIST.0001.W1FIELD1">Field 1 Value Col 2</lxml:column>  
<lxml:column name="W1LABEL2"  
id="SAMPLE\_LIST.0001.W1LABEL2">Field 2 Label Col 3</lxml:column>  
<lxml:column name="W1FIELD2"  
id="SAMPLE\_LIST.0001.W1FIELD2">Field 2 Value Col 3</lxml:column>  
<lxml:column name="W1LABEL3"  
id="SAMPLE\_LIST.0001.W1LABEL3">Field 3 Label Col 2</lxml:column>  
<lxml:column name="W1FIELD3"  
id="SAMPLE\_LIST.0001.W1FIELD3">Field 3 Value Col 2</lxml:column>  
<lxml:column name="W1LABEL4"  
id="SAMPLE\_LIST.0001.W1LABEL4">Field 4 Label Col 3</lxml:column>

```

<lxml:column name="W1FIELD4"
id="SAMPLE_LIST.0001.W1FIELD4">Field 4 Value Col 3</lxml:column>
<lxml:column name="W1LABEL5"
id="SAMPLE_LIST.0001.W1LABEL3">Field 5 Label Col 2</lxml:column>
<lxml:column name="W1FIELD5"
id="SAMPLE_LIST.0001.W1FIELD3">Field 5 Value Col 2</lxml:column>
<lxml:column name="W1LABEL6"
id="SAMPLE_LIST.0001.W1LABEL4">Field 6 Label Col 3</lxml:column>
<lxml:column name="W1FIELD6"
id="SAMPLE_LIST.0001.W1FIELD4">Field 6 Value Col 3</lxml:column>
<lxml:column name="W1IMAGE"
id="SAMPLE_LIST.0001.W1IMAGE">IMAGE_FILE</lxml:column>
<lxml:column name="W1LABEL7"
id="SAMPLE_LIST.0002.W1LABEL7">Field 7 Label
Legend</lxml:column>
<lxml:column name="W1FIELD7"
id="SAMPLE_LIST.0002.W1FIELD7">Field 7 Value
Legend</lxml:column>
<!-- <lxml:column name="W1LABEL8"
id="SAMPLE_LIST.0002.W1LABEL8">Field Label Row 8</lxml:column> -
->
</lxml:entry>
</lxml:list-entries>
</lxml:list>
</lxml:data>
<wd:template name="iii_ListRow">
<wd:description icon="icons/std_grid.ico">
<wd:name lang="ENG">Training List Row Weblet</wd:name>
</wd:description>
<wd:param name="TRN List">
<wd:tip lang="ENG">The name of the weblet.</wd:tip>
</wd:param>
<wd:param name="listname">
<wd:tip lang="ENG">The name of the list to use to populate the cells in the
grid.</wd:tip>
</wd:param>
<wd:param name="List_field_label_1">
<wd:tip lang="ENG">The name of the field in the list to use as a label in list
entry, Row 1, Col 2.</wd:tip>

```

</wd:param>  
<wd:param name="List\_field\_value\_1">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 1, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_2">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in list entry, Row 1, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_2">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 1, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_3">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in list entry, Row 2, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_3">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 2, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_4">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in list entry, Row 2, Col 3.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_4">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 2, Col 3.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_5">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in list entry, Row 3, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_5">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 3, Col 2.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_6">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in list

entry, Row 3, Col 3.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_6">  
<wd:tip lang="ENG">The name of the list field to show in list entry, Row 3, Col 3.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_label\_7">  
<wd:tip lang="ENG">The name of the field in the list to use as a label in the fieldset Legend.</wd:tip>  
</wd:param>  
<wd:param name="List\_field\_value\_7">  
<wd:tip lang="ENG">The name of the list field to show in t.</wd:tip>  
</wd:param>  
<wd:param name="even\_row\_class">  
<wd:tip lang="ENG">The Cascading Stylesheet class for even row entries in the list.</wd:tip>  
</wd:param>  
<wd:param name="odd\_row\_class">  
<wd:tip lang="ENG">The Cascading Stylesheet class for odd row entries in the list.</wd:tip>  
</wd:param>  
</wd:template>

```
<xsl:template name="iii_ListRow">
<xsl:param name="listname" wd:type="std:list_name_out" />
<xsl:param name="List_field_label_1" select=""W1LABEL1""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_1" select=""W1FIELD1""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_2" select=""W1LABEL2""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_2" select=""W1FIELD2""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_3" select=""W1LABEL3""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_3" select=""W1FIELD3""
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_4" select=""W1LABEL4""
wd:type="std:list_field_name[list=$listname]" />
```

```

<xsl:param name="List_field_value_4" select="'W1FIELD4'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_5" select="'W1LABEL5'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_5" select="'W1FIELD5'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_6" select="'W1LABEL6'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_6" select="'W1FIELD6'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_label_7" select="'W1LABEL7'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_field_value_7" select="'W1FIELD7'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="List_image_field" select="'W1IMAGE'"
wd:type="std:list_field_name[list=$listname]" />
<xsl:param name="even_row_class" select="'even_row'"
wd:type="std:css_style_class[tagName='table']" />
<xsl:param name="odd_row_class" select="'odd_row'"
wd:type="std:css_style_class[tagName='table']" />
<xsl:param name="hide_if" select="false()" wd:type="std:boolean"
wd:tip_id="" />
<input type="hidden" name="RESULTS.."
value="{count(/lxml:data/lxml:lists/lxml:list[@name=$listname]/lxml:list-
entries/lxml:entry[1])}" />
<xsl:if test="$listname != " and not($lweb_design_mode)">
<table border="0" cellspacing="0" cellpadding="0">
<tr>
<td>
<xsl:for-each
select="/lxml:data/lxml:lists/lxml:list[@name=$listname]/lxml:list-
entries/lxml:entry">
<xsl:call-template name="iii_ListRow.private">
<xsl:with-param name="listrow" select="." />
<xsl:with-param name="listname" select="$listname" />
<xsl:with-param name="List_field_label_1"
select="$List_field_label_1" />
<xsl:with-param name="List_field_value_1"
select="$List_field_value_1" />

```

```
<xsl:with-param name="List_field_label_2"
select="$List_field_label_2" />
<xsl:with-param name="List_field_value_2"
select="$List_field_value_2" />
<xsl:with-param name="List_field_label_3"
select="$List_field_label_3" />
<xsl:with-param name="List_field_value_3"
select="$List_field_value_3" />
<xsl:with-param name="List_field_label_4"
select="$List_field_label_4" />
<xsl:with-param name="List_field_value_4"
select="$List_field_value_4" />
<xsl:with-param name="List_field_label_5"
select="$List_field_label_5" />
<xsl:with-param name="List_field_value_5"
select="$List_field_value_5" />
<xsl:with-param name="List_field_label_6"
select="$List_field_label_6" />
<xsl:with-param name="List_field_value_6"
select="$List_field_value_6" />
<xsl:with-param name="List_field_label_7"
select="$List_field_label_7" />
<xsl:with-param name="List_field_value_7"
select="$List_field_value_7" />
<xsl:with-param name="List_image_field"
select="$List_image_field" />
<xsl:with-param name="even_row_class"
select="$even_row_class" />
<xsl:with-param name="odd_row_class"
select="$odd_row_class" />
<xsl:with-param name="hide_if" select="$hide_if" />
</xsl:call-template>
</xsl:for-each>
</td>
</tr>
</table>
</xsl:if>
<!-- ===== --
>
```

```
<xsl:if test="not($hide_if) or $lweb_design_mode">
<xsl:if test="$lweb_design_or_preview">
<xsl:call-template name="std_script_reference.private" />
<xsl:call-template name="std_style_reference.private">
<xsl:with-param name="caller_name" select="'std_button.xml'" />
</xsl:call-template>
</xsl:if>
<!-- ===== -->
<xsl:if test="$listname = " or $lweb_design_mode">
<table cellspacing="0" cellpadding="0" border="1">
<tbody>
<tr>
<td>
<xsl:for-each select="document(')/*//lxml:data/lxml:list/lxml:list-
entries/lxml:entry">
<xsl:call-template name="iii_ListRow.private">
<xsl:with-param name="listrow" select="." />
<xsl:with-param name="listname" select="$listname" />
<xsl:with-param name="List_field_label_1"
select="$List_field_label_1" />
<xsl:with-param name="List_field_value_1"
select="$List_field_value_1" />
<xsl:with-param name="List_field_label_2"
select="$List_field_label_2" />
<xsl:with-param name="List_field_value_2"
select="$List_field_value_2" />
<xsl:with-param name="List_field_label_3"
select="$List_field_label_3" />
<xsl:with-param name="List_field_value_3"
select="$List_field_value_3" />
<xsl:with-param name="List_field_label_4"
select="$List_field_label_4" />
<xsl:with-param name="List_field_value_4"
select="$List_field_value_4" />
<xsl:with-param name="List_field_label_5"
select="$List_field_label_5" />
<xsl:with-param name="List_field_value_5"
select="$List_field_value_5" />
<xsl:with-param name="List_field_label_6"
```

```

select="$List_field_label_6" />
<xsl:with-param name="List_field_value_6"
select="$List_field_value_6" />
<xsl:with-param name="List_image_field"
select="$List_image_field" />
<xsl:with-param name="even_row_class"
select="$even_row_class" />
<xsl:with-param name="odd_row_class"
select="$odd_row_class" />
<xsl:with-param name="hide_if" select="$hide_if" />
</xsl:call-template>
</xsl:for-each>
</td>
</tr>
</tbody>
</table>
</xsl:if>
</xsl:if>
</xsl:template>

```

```

<xsl:template name="iii_ListRow.private">
<xsl:param name="listrow" select="." />
<xsl:param name="listname" />
<xsl:param name="List_field_label_1" />
<xsl:param name="List_field_value_1" />
<xsl:param name="List_field_label_2" />
<xsl:param name="List_field_value_2" />
<xsl:param name="List_field_label_3" />
<xsl:param name="List_field_value_3" />
<xsl:param name="List_field_label_4" />
<xsl:param name="List_field_value_4" />
<xsl:param name="List_field_label_5" />
<xsl:param name="List_field_value_5" />
<xsl:param name="List_field_label_6" />
<xsl:param name="List_field_value_6" />
<xsl:param name="List_field_label_7" />
<xsl:param name="List_field_value_7" />
<xsl:param name="List_image_field" />
<xsl:param name="even_row_class" />

```

```

<xsl:param name="odd_row_class" />
<!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->
<fieldset>
<legend class="caption">
<b>
<xsl:value-of select="lxml:column[@name=$List_field_label_7]" />&#160;
</b>
<xsl:value-of select="lxml:column[@name=$List_field_value_7]" />
</legend>
<table cellspacing="0" cellpadding="0" width="750" border="0"
__evenrc="{ $seven_row_class}" __oddrcc="{ $odd_row_class}">
<xsl:attribute name="class">
<xsl:choose>
<xsl:when test="position() mod 2 = 0">
<xsl:value-of select="$seven_row_class" />
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="$odd_row_class" />
</xsl:otherwise>
</xsl:choose>
</xsl:attribute>
<tbody>
<tr>
<td style="height: 20px;" valign="middle" width="100"
align="center">
<a class="thumbnail">
<span>

<br />
</span>
</a>
</td>
<td>
<table>
<tbody>
<tr>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">

```

```

<xsl:choose>
<xsl:when test="$List_field_value_1 = 'W1LABEL1'">
<xsl:value-of select="lxml:column[@name=$List_field_label_1]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_1" />
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_1]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_value_1 = 'W1FIELD1'">
<xsl:value-of select="lxml:column[@name=$List_field_value_1]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_1"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_value_1]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_label_2 = 'W1LABEL2'">
<xsl:value-of select="lxml:column[@name=$List_field_label_2]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_2" />
</xsl:otherwise>

```

```

</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_2]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_value_2 = 'W1FIELD2'">
<xsl:value-of select="lxml:column[@name=$List_field_value_2]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_2"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_value_2]" />
</xsl:otherwise>
</xsl:choose>
</td>
</tr>
<tr>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_label_3 = 'W1LABEL3'">
<xsl:value-of select="lxml:column[@name=$List_field_label_3]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_3" />
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_3]" />

```

```
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_value_3 = 'W1FIELD3'">
<xsl:value-of select="lxml:column[@name=$List_field_value_3]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_3"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_value_3]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_label_4 = 'W1LABEL4'">
<xsl:value-of select="lxml:column[@name=$List_field_label_4]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_4" />
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_4]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
```

```

<xsl:choose>
<xsl:when test="$List_field_value_4 = 'W1FIELD4'">
<xsl:value-of select="lxml:column[@name=$List_field_value_4]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_4"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_value_4]" />
</xsl:otherwise>
</xsl:choose>
</td>
</tr>
<tr>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_label_5 = 'W1LABEL5'">
<xsl:value-of select="lxml:column[@name=$List_field_label_5]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_5" />
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_5]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_value_5 = 'W1FIELD5'">
<xsl:value-of select="lxml:column[@name=$List_field_value_5]" />
</xsl:when>

```

```

<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_5"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_value_5]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td class="caption" valign="middle" width="10%">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_label_6 = 'W1LABEL6'">
<xsl:value-of select="lxml:column[@name=$List_field_label_6]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_label_6" />
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="lxml:column[@name=$List_field_label_6]" />
</xsl:otherwise>
</xsl:choose>
</td>
<td valign="middle" width="25%" align="left">
<xsl:choose>
<xsl:when test="$lweb_design_mode">
<xsl:choose>
<xsl:when test="$List_field_value_6 = 'W1FIELD6'">
<xsl:value-of select="lxml:column[@name=$List_field_value_6]" />
</xsl:when>
<xsl:otherwise>Value of &#160;<xsl:value-of select="$List_field_value_6"
/>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:otherwise>

```

```
<xsl:value-of select="lxml:column[@name=$List_field_value_6]" />
</xsl:otherwise>
</xsl:choose>
</td>
</tr>
</tbody>
</table>
</td>
</tr>
</tbody>
</table>
</fieldset>
</xsl:template>
</xsl:transform>
```

## WAM095 - LOB Data Types and Stream Files

### Objectives

- To demonstrate how to develop a WAM with a WebRoutine, that uses the Response() parameter to output a file to the browser.

In order to complete these objectives, you will follow one of two paths:

1. If you are **using the VLF application and DXDOCS** file for employee documents you should do the following:

Step 1. Install Required Documents

Step 2. Set up Documents for an Employee

Step 3. Create WAM to Display Employee Documents

Step 4. Enhance Appearance of the Documents List (Optional)

Step 5. Set up the Documents List

Step 6. Test your Enhanced WAM

Summary

2. If you are **NOT** using the VLF application and DXDOCS file you carry out these steps:

Step 1. Install Required Documents

Step 3a. Create WAM to Display Employee Documents

Step 4. Enhance Appearance of the Documents List (Optional)

Step 5a. Set up the Documents List

Step 6. Test your Enhanced WAM

Summary

### Before You Begin

WAMs allow you to serve stream files that:

- You don't want to store on your Web server
- Documents whose contents are stored in your application data base.
- Documents that are created on demand.

You can serve the contents of LOB data types (BLOBs and CLOBs) and stream files located in your Application Server with special WebRoutines.

```
Webroutine Name(SEND_SAMPLE) Response(#HTTPR) Desc('Sample Document')  
  * MYPATH is the directory where the sample documents are stored  
  #HTTPR.ContentFile := #MYPATH + 'sample'  
Endroutine
```

The WebRoutine can contain RDMLX code to create the contents of the file or determine which file to send. The only requirement is that you set the ContentFile to the file name that you want to serve.

## Step 1. Install Required Documents

In this exercise you will need some files from the zip file described in *Before you Begin* section at the start of [WAM Tutorials](#).

1. You will need the following files to complete this exercise:

V\_Brown\_CV.pdf

Employee\_Confidentiality\_Agreement.pdf

A0070\_Details.doc

A0070\_Holidays\_2013.xls

A0070\_Notes.txt

A0070\_Presentation.ppt

MYDOCS.txt

If you are not using the VLF application to store images in file DXDOCS, the file MYDOCS.txt contains a list of the files to be displayed, which will be read by your WAM.

Copy these files from the Extra Files to the folder: C:\Program Files (x86)\LANSA\

### IBM i

- a. If you are running your WAMs on the IBM i, copy the file MYDOCS.txt into an IFS folder such as /LANSA\_d13pgmlib/ where d13pgmlib is your LANSA program library.
- b. Copy all other files into the IFS folder, such as:  
/LANSA\_d13pgmlib/webserver/images/
- c. Change your RDML code to use the first path to TRANSFORM\_FILE. Use the second path to retrieve the files to send to the web server.

[WAM095. Appendix A](#) contains sample code for both Windows and IBM i execution.

Alternatively, use any files that you have available that will be recognized by windows and can be opened in the browser.

**Note:** In this case you will need to create a simple text file MYDOCS.txt containing a list of the files you wish to display.

2. A later step requires a set of small gif images for each file type, which will be displayed as a clickable image in the first column of the employee documents list.

## Windows

These files are supplied in the Extra Files zip file. Copy the files from the File\_GIFS folder to:

c:\Program Files (x86)\LANSA\WebServer\Images

doc.gif	pdf.gif	txt.gif
htm.gif	ppt.gif	xls.gif
html.gif	text.gif	

## IBM i

If you are running your WAM on the IBM i, copy the gif files to the IFS for example to:

/LANSA\_d13pgmlib/webserver/images/ where d13pgmlib is the name of your program library.

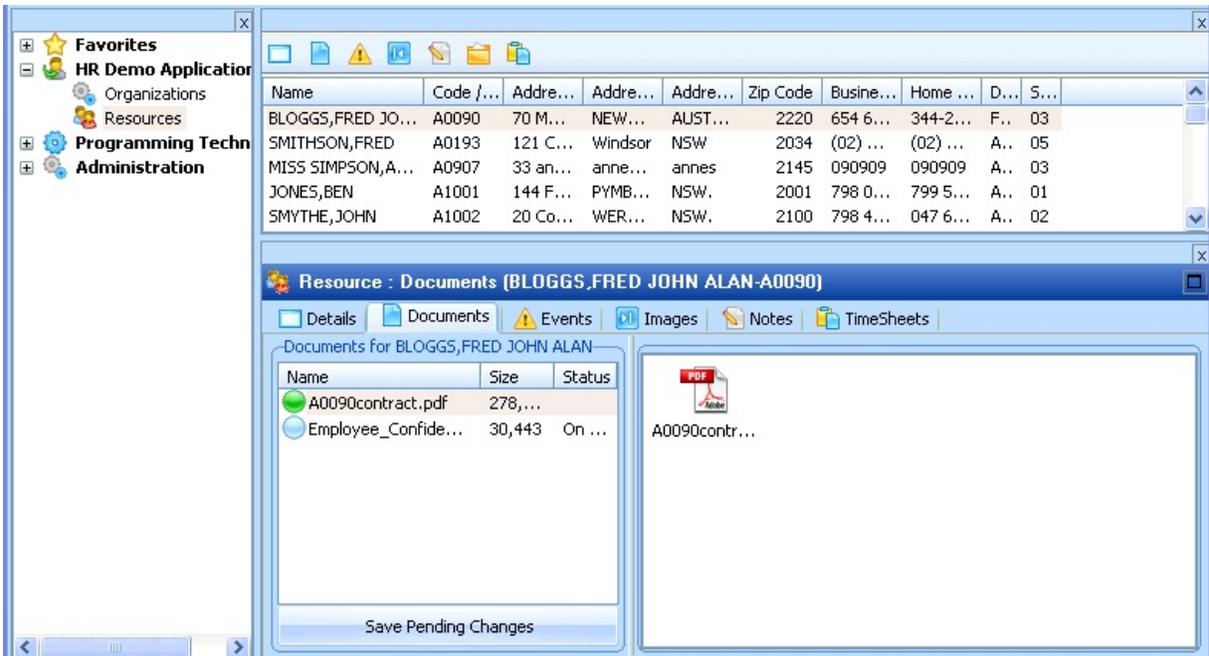
## Scenario

This exercise may be completed in two ways:

1. Setting up documents for an employee in the file DXDOCS using the demonstration Visual Frameworks application. Continue immediately below to follow this approach
2. Reading an employee record and setting up a working list containing the names of images associated with the selected employee. To follow this approach, go directly to [Step 3a. Create WAM to Display Employee Documents](#).

**The following assumes that you have Visual Frameworks (VLF) installed in the partition being used for the WAM training tutorials.**

The file DXDOCS is maintained by the *Documents* command handler in the shipped VLF *HR Demo Application* for the business object, *Resources*.



You will use this application to set up documents for at least one employee. Your WAM will then enable these documents to be displayed in the browser. If you are testing your WAM applications locally (on your development PC), you can start the Visual Framework and work offline. You will be updating the DXDOCS file locally.

If you are creating WAM applications to run on an IBM i server, your Visual Framework must be configured to logon to the IBM i server. Your documents will then be stored in the file DXDOCS on the server.

You can find more details on how the Document command handler works (reusable part DX\_DOCS) in the comments included in the component source RDML.

The WAM you will build in this exercise, iiiDspEmpDocs – Display Employee Documents, is a simple WAM that displays details for a single employee and a list of associated documents retrieved from file DXDOCS. When a document is selected it will be displayed in a new browser window.

## File DXDOCS

Review the DXDOCS file definition in the *Repository*. Note that the file keys include all possible VLF instance list key fields. The highest level key contains the business object name. In this case, business object name will be DEM\_ORG\_SEC\_EMP. See the Visual LANSA Framework properties sheet for the *Resources* business object. The lowest level file key contains the document file name. For the business object *Resources*, the file key will be:

<b>Field</b>	<b>Value</b>
DF_ELOID	<b>DEM_ORG_SEC_EMP</b>
DF_ELKEY1	<b>#DEPARTMENT</b>
DF_ELKEY2	<b>#SECTION</b>
DF_ELKEY3	<b>#EMPNO</b>
DF_ELFNAM	<b>File Name</b>

All keys not shown in the table will be blank.

## **Understanding BLOBs**

Refer to the *Technical Guide* for a detailed explanation of how to use the BLOB data type.

BLOB is a variable-length binary field of undefined maximum length.

The most common operation with BLOBs are saving files into the database and retrieving them so they can be viewed/edited/etc. In RDMLX, BLOB fields are manipulated as filenames.

Following is an example of saving a JPG as a BLOB field in a database file:

```
#MYBLOB := 'C:\temp\mypicture.jpg'
UPDATE FIELDS(#MYBLOB) IN_FILE(FILE1)
```

In RDMLX when a BLOB or CLOB field is used, keep in mind that the field contains a filename, not the actual data in the object. In RDMLX, LANSA LOB fields will be manipulated as filenames. It is only in database IO commands that the BLOB or CLOB actual data itself is handled by reading from or writing to the named file.

Rather than the default property **.Value**, fields of type BLOB have a default property called **.FileName** to clearly indicate that changing the "value" of the field is actually changing its default property which is a file name property.

When BLOB and CLOB data is read from the database, files are automatically created in the directory structures under the LPTH= directory (for more information, refer to [Standard X\\_RUN Parameters](#) in the *Technical Reference*

*Guide).*

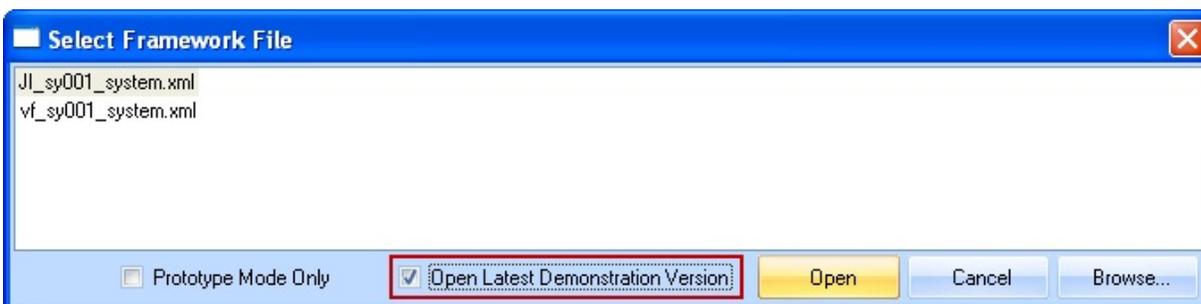
You can use the VLF Documents command handler to store any type of document for an employee. For example: PDF, DOC, XLS, TXT etc.

## Step 2. Set up Documents for an Employee

### Before You Begin

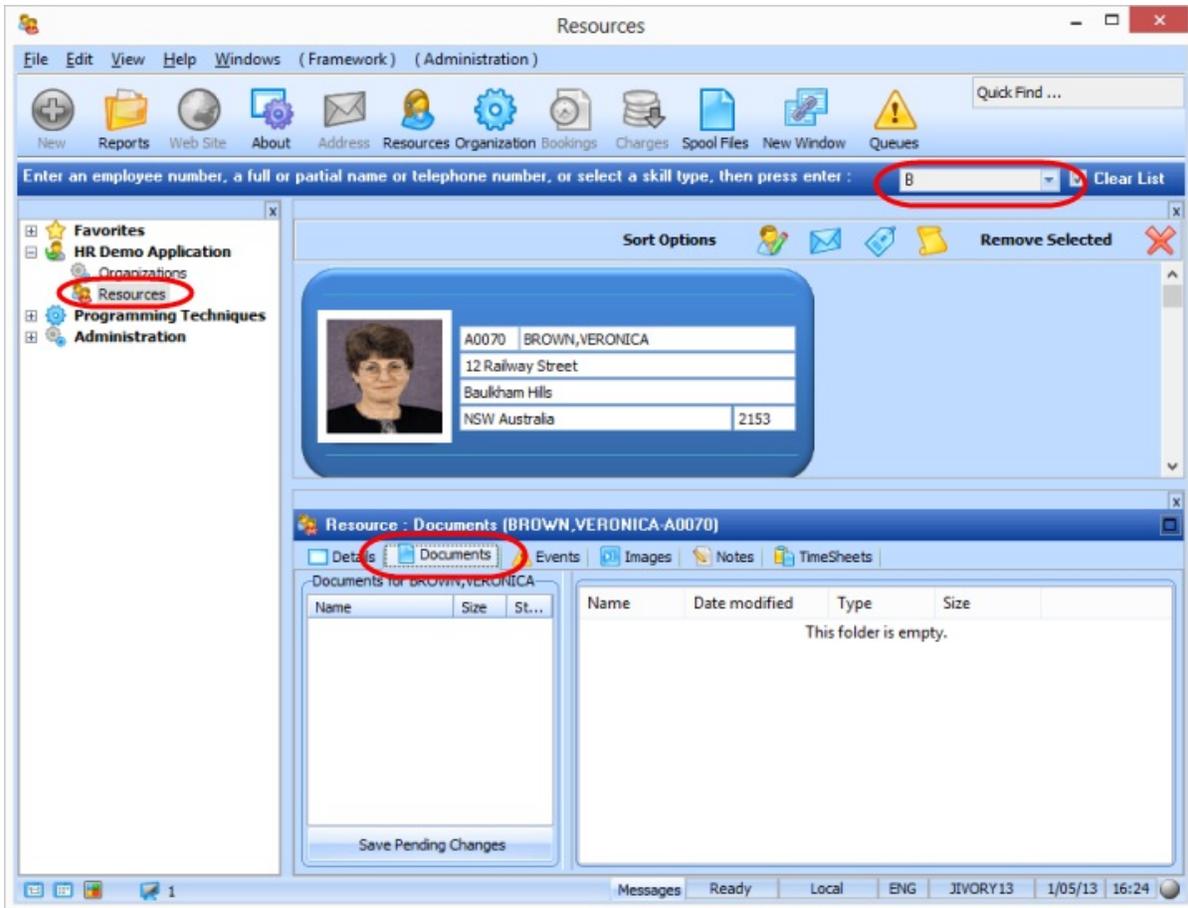
1. Start the Visual LANSa Framework (VLF) from the *VL Framework* group on the *Tools* ribbon.

If this is the first time VLF has been run, the shipped framework (vf\_sy0001\_system.xml) will be run by default. If necessary, in the following dialog select the *Open Latest Demonstration Version* checkbox to select the latest shipped framework:

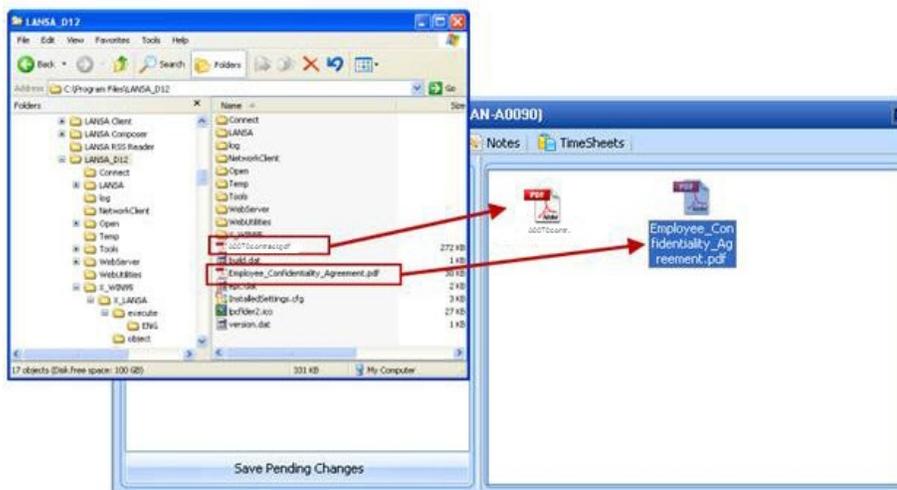


2. Select the *HR Demo Application* and then select the *Resources* business object. Enter **B** in the mini filter in the toolbar and press *Enter* to display employees in the Instance list. Select employee BROWN, VERONICA, A0070.

Select the *Documents* tab for this employee.



3. Open *Windows Explorer* and navigate to folder `c:\Program Files\LANSA`. Drag the sample files (types: doc, txt, ppt, xls and pdf) into the right hand *Documents* panel as shown:



4. Select the *Save Pending Changes* button to save the documents to the file

DXDOCS.

### Step 3. Create WAM to Display Employee Documents

1. Create a new WAM:

Name: **iiiDspEmpDocs**

Description: **Display Employee Documents**

Layout Weblet: **iiilay01**

2. Define your WAM based on the following logic:

- Map field STDREENTRY for \*both as a \*hidden field
- Define a Group\_by, EMPDATA for fields EMPNO, SURNAME, GIVENAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, DEPARTMENT and SECTION.
- Define a working list DOCLIST, containing fields DF\_ELFNAM and PRIFILRRN.

PRIFILRRN should be a hidden field.

- Define a WebRoutine BEGIN

Map field EMPNO for \*both

Map fields FULLNAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, DEPARTMENT and SECTION and list DOCLIST for \*output. These fields should have a display attribute of \*output.

If STDREENTRY = S

- Clear list DOCLIST
- Fetch fields for group\_by EMPDATA from file PSLMST with the key EMPNO
- FULLNAME = GIVENAME + ', ' + SURNAME
- Select field DF\_ELFNAM from the file DXDOCS with the key 'DEM\_ORG\_SEC\_EMP', DEPARTMENT, SECTION, EMPNO and return relative record number to PRIFILRRN
- Add entry to DOCLIST
- End of select

End of if

- End of routine
- Define a WebRoutine SEND\_DOCUMENT with a Response keyword of

#HTTPR

- Map for \*input field PRIFILRRN
- Fetch field DX\_ELLOB from the file DXDOCS with the relative record number PRIFILRRN.
- Set #HTTPR property ContentFile to #DXELLOB.FileName
- End of routine

**Note:** The field DX\_ELLOB returns the file into a local temporary directory, using an 8.3 filename. For example:

C:\DOCUME~1\pcuser\LOCALS~1\Temp\lobuser\pcuser\5152\dxdoc

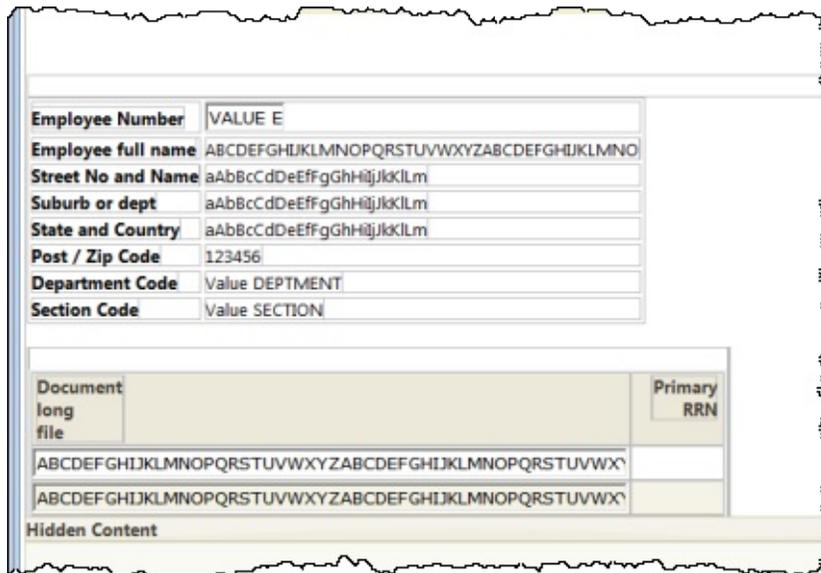
This path may be defined by the LPTH=directory run time parameter. If the LPTH parameter is not defined, the path used will be the TPTH setting. The default value for this path is the user's temporary path, for example:

C:\Users\John\AppData\Local\Temp

This can be quickly found using %temp% in the Explorer address bar.

For further information, refer to the [Standard X\\_RUN Parameters](#) in the *Technical Reference Guide*.

3. Compile **iiiDspEmpDocs** and open the WebRoutine **BEGIN** in the *Design* view. It should look like the following:



4. Drop a *Push Button* weblet alongside the employee number field. Set up the push button properties:

--	--

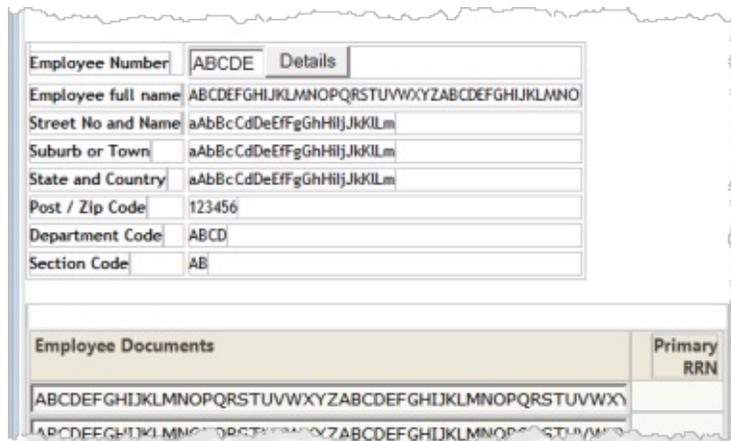
Property	Value
Caption	<b>Details</b>
On_click_wname	<b>BEGIN</b>
submitExtraFields	<b>Field Name: STDREENTRY</b>
	<b>Literal value: S</b>

5. Save your changes.
6. Select the column heading "Document long file" and delete it.  
Type in a new the column heading "Employee Documents".

You may need to click somewhere else in the layout, to refresh the column heading with your changes.

7. Save your changes.

Your design should look like the following:



8. Drop an *Anchor* weblet into the file name column of the list (the left hand column). Ignore the increase in width of this column. At run time it will display with the width of the actual file names.

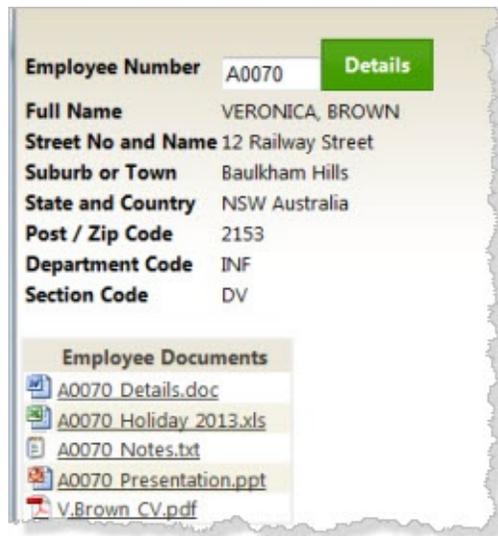
Set up the *Anchor* weblet properties:

Property	Value
Currentrowhfield	<b>PRIFILRRN</b>

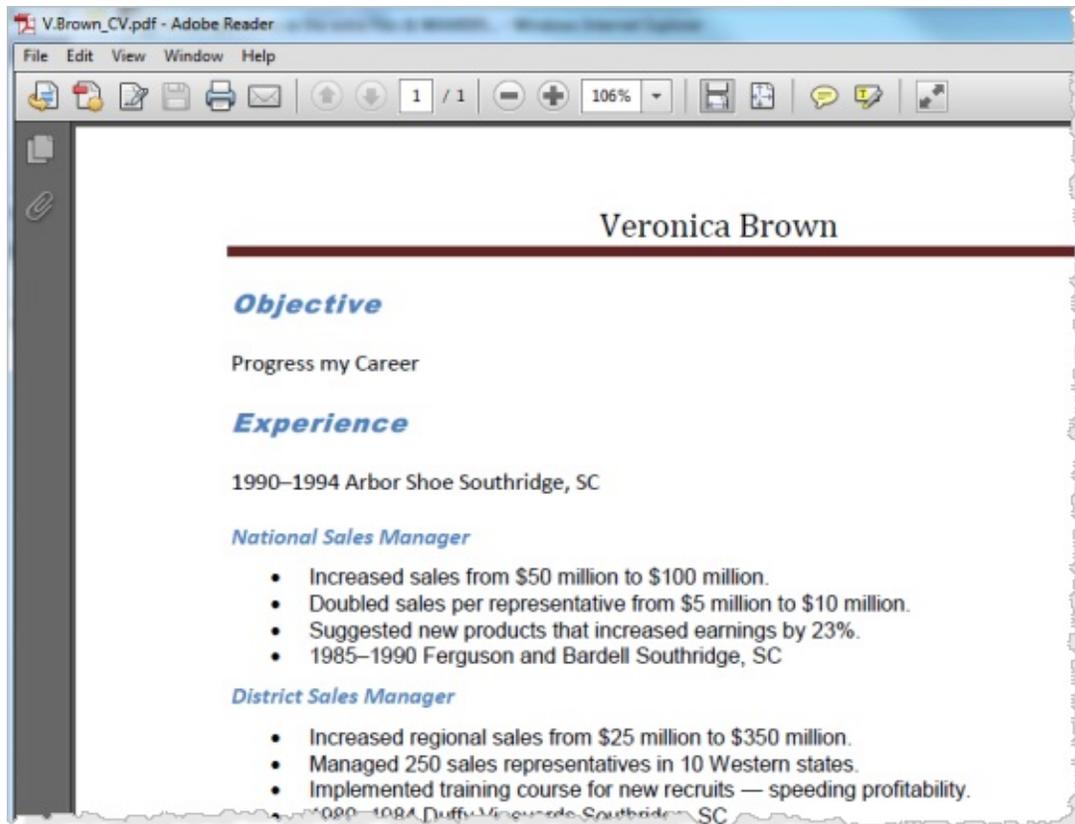
Currentrownumval	<b>\$PRIFILRRN</b>
On_click_wrname	<b>SEND_DOCUMENT</b>
Show_in_new_window	<b>True</b>

Save your changes.

9. Execute and test your WAM in the browser.
  - a. Enter employee number A0070 and select the *Details* push button. Your web page should look like the following:



- b. Select one of the documents shown in the list. The document should be displayed in a new browser window.



Continue at [Step 4. Enhance Appearance of the Documents List \(Optional\)](#)

## Step 3a. Create WAM to Display Employee Documents

Follow this step if you did not set up documents for an employee using the demonstration VLF application, in the file DXDOCS.

This WAM will display a fixed list of documents for an employee, based on a data from a supplied text file. To simplify building the list of documents, the WAM reads a text file MYDOCS.txt to populate a working list. Another working list is populated for display. A later step will add another column to this list which will contain a clickable image.

1. Create a new WAM:

*Name:* **iiiDspEmpDocs**

*Description:* **Display Employee Documents**

*Layout Weblet:* **iiilay01**

2. Create your WAM based on the following logic:

- Map field STDREENTRY for \*both as a \*hidden field
- Define a Group\_By, EMPDATA for fields, EMPNO, SURNAME, GIVENAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, DEPARTMENT and SECTION.
- Define a working list DOCLIST containing fields DF\_ELFNAM
- Define a working list MYDOCS containing field STD\_TEXTL
- Define a work field RETCODE with reference field IO\$STS
- Define a webroutine BEGIN
- Map field EMPNO for \*both
- Map fields FULLNAME, ADDRESS1, ADDRESS2, ADDRESS3, POSTCODE, DEPARTMENT and SECTION for \*output. All fields should have a display attribute of \*output
- If STDREENTRY = S
- Clear list DOCLIST
- Clear list MYDOCS
- Fetch group\_by EMPDATA from file PSLMST with the key EMPNO
- Use the BIF transform\_file to populate the list MYDOCS from the file "C:\Program Files (x86)\LANSA\MYDOCS.txt"
- Selectlist MYDOCS

- Change DF\_ELFNAM to STD\_TEXTL
  - Add an entry to DOCLIST
  - Endselect
  - Endif
3. Define a WebRoutine SEND\_DOCUMENT with a Response keyword of #HTTTPR
- Map field DF\_ELFNAM for \*input
  - Set #HTTTPR, property ContentFile to "C:\Program Files (x86)\LANSA\" + DF\_ELFNAM
  - Endroutine

Sample code is supplied in [WAM095. Appendix B.](#)

4. Compile your WAM. Open the WebRoutine **BEGIN** in the *Design* view, it should look like the following:

5. Drop a *Push Button* weblet alongside the employee number field. Set up the push button properties:

Property	Value
Caption	Details
On_Click_wrname	BEGIN
submitExtraFields	STDREENTRY

Literal Value: S
------------------

6. Save your changes.
7. Select the column heading "Document long file" and delete it.  
Type in a new column heading - **Employee Documents**.  
You may need to click on another part of the layout to refresh the column heading.
8. Drop an *Anchor* weblet into the list's file name column. Ignore the increase in width of the column. At run time it will be displayed with the width of the actual file names.

Set up the Anchor weblet properties:

<b>Property</b>	<b>Value</b>
Currentrowhfield	DF_ELFNAM
Currentrownumval	\$DF_ELFNAM
On_click_wname	SEND_DOCUMENT
Show_in_new_window	True

Your design should look like the following:

Employee Number	ABCDE	Details
Employee full name	ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNO	
Street No and Name	aAbBcCdDeEfFgGhHijJkKlLm	
Suburb or Town	aAbBcCdDeEfFgGhHijJkKlLm	
State and Country	aAbBcCdDeEfFgGhHijJkKlLm	
Post / Zip Code	123456	
Department Code	ABCD	
Section Code	AB	
<b>Employee Documents</b>		
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOQRSTUVWXYZABCDEFGHI		
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOQRSTUVWXYZABCDEFGHI		

9. Save your changes.

10. Execute your WAM in the browser. Any employee number may be entered (A0070, A0090, A1234 etc). The WAM will always display the employee details and a fixed list of documents.

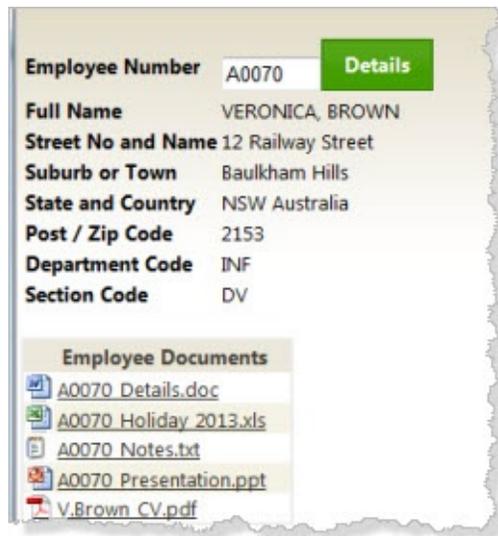
Click on one of the documents to display it. The way that the browser and Windows handles the request will depend on the version of browser and the version of Windows being used. For example, in Windows 7 with MS Office installed and IE10, the txt and pdf files are displayed in a new browser tab.

The Office documents prompt to be displayed in the required Office program.

Continue with [Step 4. Enhance Appearance of the Documents List \(Optional\)](#)

## Step 4. Enhance Appearance of the Documents List (Optional)

This step adds a column containing a suitable image to the list of documents, as shown:



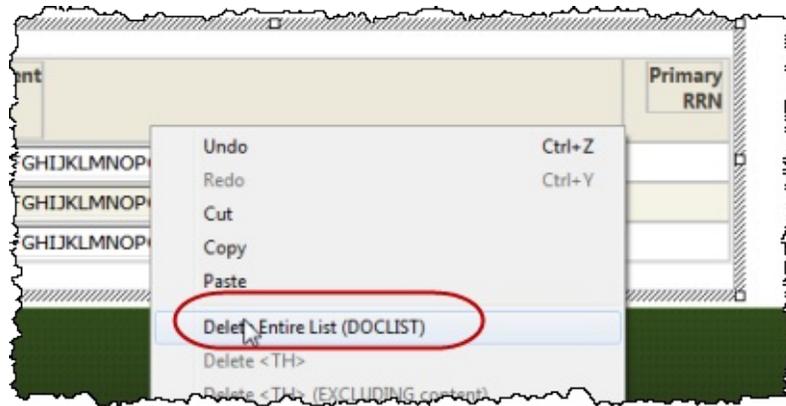
1. Extend WAM **iiiDspEmpDocs** as follows:
  - a. Define a character work field FILENAME, length 8.
  - b. Add field FILENAME as the first field in working list DOCLIST.
2. The field FILENAME needs to be populated with the name of the appropriate image to display for each BLOB, depending on its file type. The field DF\_ELFNAM already contains the long file name for each BLOB (document).

Extend the **BEGIN** WebRoutine as shown. Within the SELECT loop add the following logic, before the ADD\_ENTRY:

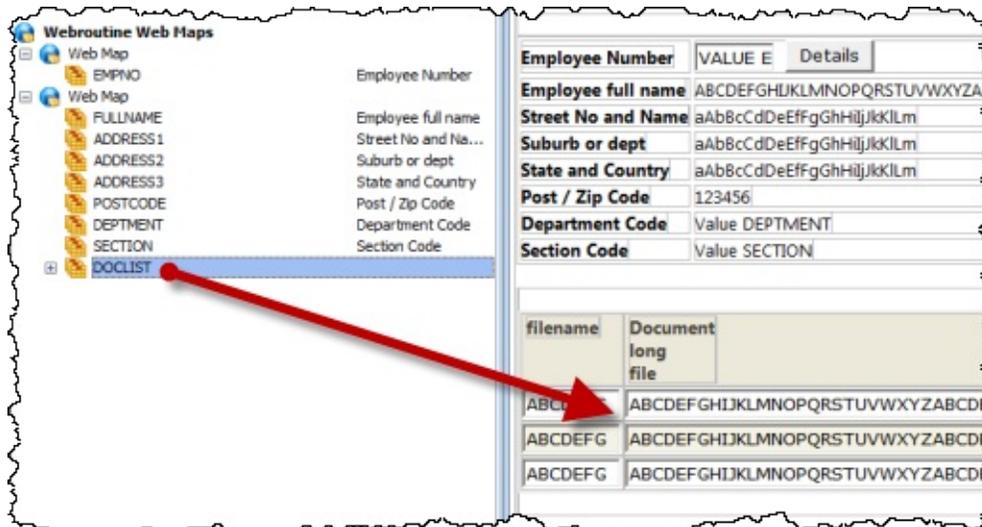
```
#std_count := #df_elfnam.LastPositionOf( '.' )  
#std_texts := #df_elfnam.substring( (#std_count + 1), 4 )  
#filename := #std_texts.trim + '.gif'
```

See [WAM095. Appendix C](#) for the changed code, if required.

3. Recompile your WAM.
4. Open the WebRoutine **BEGIN** in the *Design* view. Select the **Employee Documents** list and use the context menu to *Delete Entire List*.



5. Select the *WebRoutine Output* tab and drag the list, **DOCLIST**, back onto the page.



6. Delete the column heading text for Filename. Delete the column heading, **Document long file** and replace it with **Employee Documents**.

If you are not using the VLF and the DXDOCS file, continue at [Step 5a. Set up the Documents List](#) otherwise continue at [Step 5. Set up the Documents List](#).

## Step 5. Set up the Documents List

Follow this step when the DXDOCS file is being used otherwise go to [Step 5a. Set up the Documents List](#).

1. Drag and drop a *Clickable Image* weblet into the first column (filename) and set up the weblet as follows:

<b>Property</b>	<b>Value</b>
Currentrowhfield	PRIFILRRN
Currentrownumvalue	\$PRIFILRRN
Rentryvalue	S
Tooltip	Select image to display document
on_click_wname	SEND_DOCUMENT
show_in_new_window	True
relative_image_path	\$FILENAME

2. Drag and drop an *Anchor* weblet into the second column (DF\_ELFNAM) and set up the weblet as follows:

<b>Property</b>	<b>Value</b>
currentrowhfield	PRIFILRRN
currentrownumvalue	\$PRIFILRRN
rentryvalue	S
on_click_wname	SEND_DOCUMENT
show_in_new_window	True

3. Save your changes

## Step 5a. Set up the Documents List

Follow this step when you are **NOT** using the DXDOCS file.

1. Drag and drop a *Clickable Image* weblet into the first column (filename) and set up the weblet as follows:

Property	Value
currentrowhfield	DF_ELFNAM
currentrownumvalue	\$DF_ELFNAM
rentryvalue	S
tooltip	Select image to display document
on_click_wrname	SEND_DOCUMENT
show_in_new_window	True
relative_image_path	\$FILENAME

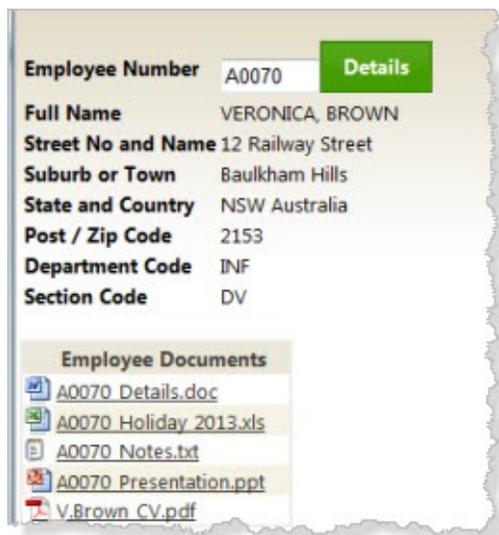
2. Drag and drop an *Anchor* weblet into the second column (DF\_ELFNAM) and set up the weblet as follows:

Property	Value
currentrowhfield	DF_ELFNAM
currentrownumvalue	\$DF_ELFNAM
rentryvalue	S
on_click_wrname	SEND_DOCUMENT
show_in_new_window	True

3. Save your changes.

## Step 6. Test your Enhanced WAM

1. Test your application for employee A0070. If you have saved other types of document for this employee, your results will look like this:



If you are not using the DXDOCS file, your WAM outputs a fixed list of documents for an employee.

2. You should now be able to click on the hyperlink in the Employee Documents column or the clickable image in the first column to display a document.

## Summary

### Important Observations

- Standard Windows documents such as Adobe Acrobat (PDF), Word (DOC) and Excel (XLS) can easily be displayed by a WAM WebRoutine using the Response(#HTTTPR) parameter.
- You should read the relevant sections in the *Web Application Module Guide* and *Technical Reference* before implementing files storing BLOB and CLOB data.

### What I Should Know

- How to write a WebRoutine that is using the Response() parameter to output a document.
- How to set up a list column with dynamic images

## **WAM095. Appendix A**

### **Using DXDOCS File - Sample RDMLX for iiiDspEmpDocs**

**This WAM can be run locally or on IBM i.**

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME #A

Def_List Name(#doclist) Fields(#df_elfnam (#prifilrrn *hidden)) Type(*Worki

Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2 *
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Select Fields(#df_elfnam) From_File(dxdocs) With_Key('DEM_ORG_SEC_E
Add_Entry To_List(#doclist)
Endselect
Endif
Endroutine

Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample I

Web_Map For(*input) Fields(#PRIFILRRN)

Fetch Fields(#df_elfnam #dx_elblob) From_File(dxdocs) With_Rrn(#prifilrrn)
#HTTPR.ContentFile := #DX_ELLOB.FileName

Endroutine

End_Com
```

## **WAM095. Appendix B**

### **Sample RDMLX for iiiDspEmpDocs, when not using DXDOCS file.**

#### **This WAM to be run locally**

```
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #deptment #section)
```

```
Def_List Name(#doclist) Fields(#df_elfnam) Type(*Working)
Def_List Name(#mydocs) Fields(#std_textl) Type(*working)
```

```
Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2
*out) (#address3 *out) (#postcode *out) (#deptment *out) (#section *out)
#doclist)
Define Field(#retcode) Reffld(#io$sts)
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Use Builtin(transform_file) With_Args(#mydocs 'c:\Program Files
(x86)\LANSA13\WAM095_DOCS.txt') To_Get(#retcode)
Selectlist Named(#mydocs)
#df_elfnam := #std_textl.trim
Add_Entry To_List(#doclist)
Endselect
Endif
Endroutine
```

```
Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample
Document')
Web_Map For(*input) Fields(#DF_ELFNAM)
#HTTPR.ContentFile := "C:\Program Files (x86)\LANSA13\" +
#DF_ELFNAM
Endroutine
```

End\_Com

**This WAM to be run on the IBM i**

```
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #deptment #section)
Def_List Name(#doclist) Fields(#df_elfnam) Type(*Working)
Def_List Name(#mydocs) Fields(#std_textl) Type(*working)
Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2
*out) (#address3 *out) (#postcode *out) (#deptment *out) (#section *out)
#doclist)
Define Field(#retcode) Reffld(#io$sts)
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Use Builtin(transform_file) With_Args(#mydocs
'/LANSA_d13pgmlib/MYDOCS.txt') To_Get(#retcode)

Selectlist Named(#mydocs)
#df_elfnam := #std_textl.trim
Add_Entry To_List(#doclist)
Endselect
Endif
Endroutine
Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample
Document')
Web_Map For(*input) Fields(#DF_ELFNAM)
#HTTPR.ContentFile := "/LANSA_d13pgmlib/webserver/images/" +
#DF_ELFNAM

Endroutine
End_Com
```

## WAM095. Appendix C

### Sample RDMLX for Enhanced Documents List

#### Using file DXDOCS

Changes are highlighted in red italics.

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #deptment #section)
```

```
Def_List Name(#doclist) Fields(#filename #df_elfnam (#prifilrrn *hidden))
Type(*Working)
```

```
Define Field(#filename) Type(*char) Length(8)
```

```
Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2
*out) (#address3 *out) (#postcode *out) (#deptment *out) (#section *out)
#doclist)
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Select Fields(#df_elfnam) From_File(dxdocs)
With_Key('DEM_ORG_SEC_EMP' #deptment #section #empno)
Return_Rrn(#PRIFILRRN)
#std_count := #df_elfnam.LastPositionOf( '.' )
#std_texts := #df_elfnam.substring( (#std_count + 1), 4 )
#filename := #std_texts.trim + '.gif'
Add_Entry To_List(#doclist)
Endselect
Endif
Endroutine
```

```
Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample
Document')
Web_Map For(*input) Fields(#PRIFILRRN)
```

```
Fetch Fields(#df_elfnam #dx_elblob) From_File(dxdocs) With_Rrn(#prifilrn)
#HTTPR.ContentFile := #DX_ELLOB.FileName
Endroutine
End_Com
```

## Not Using DXDOCS File

### WAM running locally

Changes are highlighted in red italics.

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM) Layoutweblet('iiilay01')
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #deptment #section)
```

```
Def_List Name(#doclist) Fields(#filename #df_elfnam) Type(*Working)
Def_List Name(#mydocs) Fields(#std_textl) Type(*working)
```

```
Define Field(#filename) Type(*char) Length(8)
Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2
*out) (#address3 *out) (#postcode *out) (#deptment *out) (#section *out)
#doclist)
Define Field(#retcode) Reffld(#io$sts)
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Use Builtin(transform_file) With_Args(#mydocs 'c:\Program Files
(x86)\LANSA13\MYDOCS.txt') To_Get(#retcode)
Selectlist Named(#mydocs)
#df_elfnam := #std_textl.trim
#std_count := #df_elfnam.LastPositionOf(' ')
#std_texts := #df_elfnam.substring( (#std_count + 1), 4 )
#filename := #std_texts.trim + '.gif'
```

```
Add_Entry To_List(#doclist)
```

```
Endselect
Endif
Endroutine
```

```
Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample
Document')
Web_Map For(*input) Fields(#DF_ELFNAM)
#HTTPR.ContentFile := "C:\Program Files (x86)\LANSA13\" +
#DF_ELFNAM
Endroutine
End_Com
```

## WAM running on IBM i

```
Function Options(*DIRECT)
Begin_Com Role(*EXTENDS #PRIM_WAM)
Web_Map For(*BOTH) Fields((#stdrentry *hidden))
Group_By Name(#empdata) Fields(#EMPNO #SURNAME #GIVENAME
#ADDRESS1 #ADDRESS2 #ADDRESS3 #POSTCODE #deptment #section)

Def_List Name(#doclist) Fields(#filename #df_elfnam (#prifilrn *hidden))
Type(*Working)
Define Field(#filename) Type(*char) Length(8)
Webroutine Name(Begin)
Web_Map For(*both) Fields(#empno)
Web_Map For(*output) Fields((#fullname *out) (#address1 *out) (#address2
*out) (#address3 *out) (#postcode *out) (#deptment *out) (#section *out)
#doclist)
If (#stdrentry = S)
Clr_List Named(#doclist)
Fetch Fields(#empdata) From_File(pslmst) With_Key(#empno)
#fullname := #givename + ', ' + #surname
Select Fields(#df_elfnam) From_File(dxdocs)
With_Key('DEM_ORG_SEC_EMP' #deptment #section #empno)
Return_Rrn(#PRIFILRRN)
#std_count := #df_elfnam.LastPositionOf( '.' )
#std_texts := #df_elfnam.substring( (#std_count + 1), 4 )
#filename := #std_texts.trim + '.gif'
```

```
Add_Entry To_List(#doclist)
Endselect
Endif
Endroutine
```

```
Webroutine Name(SEND_DOCUMENT) Response(#HTTPR) Desc('Sample
Document')
Web_Map For(*input) Fields(#PRIFILRRN)
Fetch Fields(#df_elfnam #dx_elblob) From_File(dxdocs) With_Rrn(#prifilrrn)
#HTTPR.ContentFile := #DX_ELBLOB.FileName
Endroutine
End_Com
```

## WAM100 - Using Cascading Style Sheets

### Objectives

To learn how your own style sheet can be set up and used to control the appearance of specific elements of your web pages.

During this exercise you will use: **Microsoft Internet Explorer, Developer Tools** (included in Internet Explorer V8, V9 and V10) that enable the structure of a web page and its style rules to be explored in detail.

In order to complete this exercise you will complete the following:

Review [What are Cascading Style Sheets?](#) and [What CSS files are loaded and how do I add my own?](#)

Step 1. [Create WAM iii Using CSS](#)

Step 2. [Create new Style Sheet](#)

Step 3. [Create an External Resource](#)

Step 4. [Apply Style Sheet to WAM iii Using CSS](#)

Step 5. [Apply External Resource to the Common Layout](#)

Step 6. [Make the Style Sheet specific to lists named EMPLIST](#)

Step 7. [Highlight a Column](#)

[Summary](#)

### Before You Begin

This exercise does not depend on knowledge gained from all preceding exercises. The following exercises should have been completed:

- [WAM005 - Create Your First WAM](#)
- [WAM010 - Using WEB\\_MAPs](#)
- [WAM015 - Working Lists](#)
- [WAM020 - WAM Navigation](#)
- [WAM030 - Employee Enquiry](#)

### What are Cascading Style Sheets?

A Cascading Style Sheet tells the browser how to display page elements. Cascading Style Sheet information determines things like the fonts and color schemes, DHTML effects, alignment, border size and color, but may also be used to define images and other features related to the interface. These properties can be assigned to individual elements identified by an ID, or groups

of elements identified by type, location and class.

Many of the shipped weblots include style (or class) properties. The default style applied to a property, and the full set of styles available in the dropdown list associated with these properties, relate directly back to the CSS file referenced on the WAM's related layout.

For full information see <http://www.w3schools.com/css>

Style sheet files (CSS) are simply text and can be edited with any text editor such as Notepad. If you are working with style sheets it's a good idea to obtain a specialist style sheet editor. There are a number available, *TopStyle* is one example. They will make your editing faster and more accurate and make it easier to navigate through and manage a large number of styles defined in a style sheet. They will also help you to more rapidly learn the options available when defining styles.

## **What CSS files are loaded and how do I add my own?**

The `std_style_v2` weblot takes care of creating all the `<link>` tags needed to load the CSS files so you need to include it in the `<head>` section of your layouts.

The `std_style_v2` weblot always loads `std_style.min.css` into every layout. This defines the non-theme related properties of all LANSA supplied weblots.

It then loads any CSS files defined by its `theme_css_filename` and `css_files` properties. These properties are provided for backwards compatibility with layouts built with older versions of the weblot. For new layouts, you should specify 'none' in `theme_css_filename` and use External Resources to define additional CSS files you want to include.

Next, it adds any CSS files defined as *External Resources* referenced in the webroutine, layout or weblots used by the webroutine.

Finally, the `std_style_v2` weblot loads a stylesheet defined by the variable `$lweb_std_css_language_overlay`. This variable is defined in the [std\\_locale](#) weblot and provides a means to apply language specific CSS modifications.

For more detail on this topic see the *Web Application Modules* guide.

## **What Cascading Style Sheets are available?**

The main CSS stylesheets are in the main style directory under the images directory.

The themed CCS stylesheets are under the jQuery subdirectory—under a subdirectory named after the theme.

See the Web Application Modules guide for a full list of the other stylesheets

available to WAMs.

## Cascading Style Sheets in Action

For an appreciation of how the shipped style sheets combine, this image is the begin WebRoutine in WAM *iiiSecMaint – Section Maintenance* with the themelet stylesheets removed:



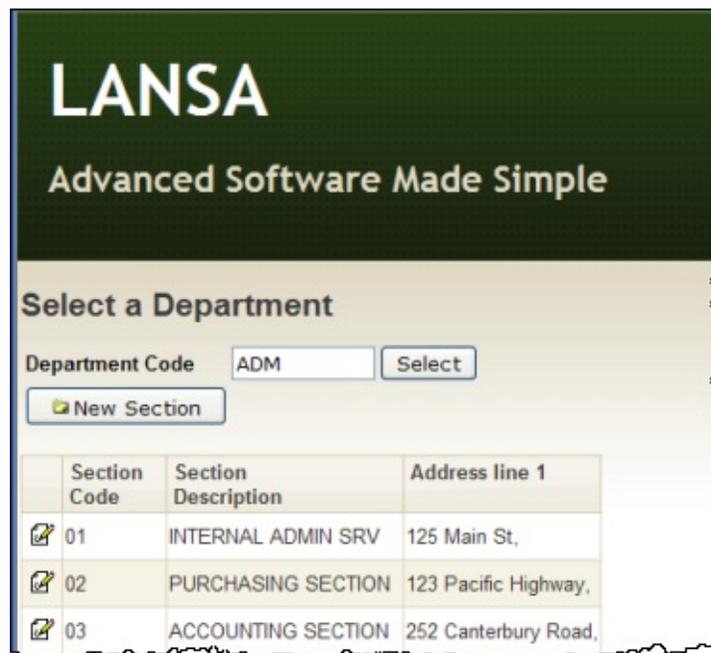
LANSA  
Advanced Software Made Simple  
[Home](#) [Services](#) [Contact](#) [About](#)

### Select a Department

Department Code

Section Code	Section Description	Address line 1
01	INTERNAL ADMIN SRV	125 Main St,
02	PURCHASING SECTION	123 Pacific Highway,
03	ACCOUNTING SECTION	252 Canterbury Road,

This image is the **begin** WebRoutine with the *SouthStreet theme* style sheets applied:



**LANSA**  
Advanced Software Made Simple

### Select a Department

Department Code

Section Code	Section Description	Address line 1
01	INTERNAL ADMIN SRV	125 Main St,
02	PURCHASING SECTION	123 Pacific Highway,
03	ACCOUNTING SECTION	252 Canterbury Road,

Clearly, to change the appearance of any element on your web page, you need to

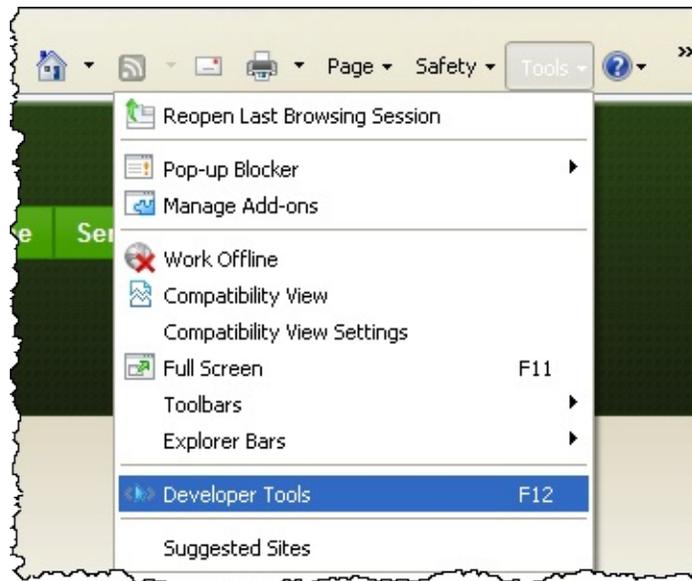
define a style sheet that overrides the styles defined by your chosen theme.

## Use CSS with Lists and Grids

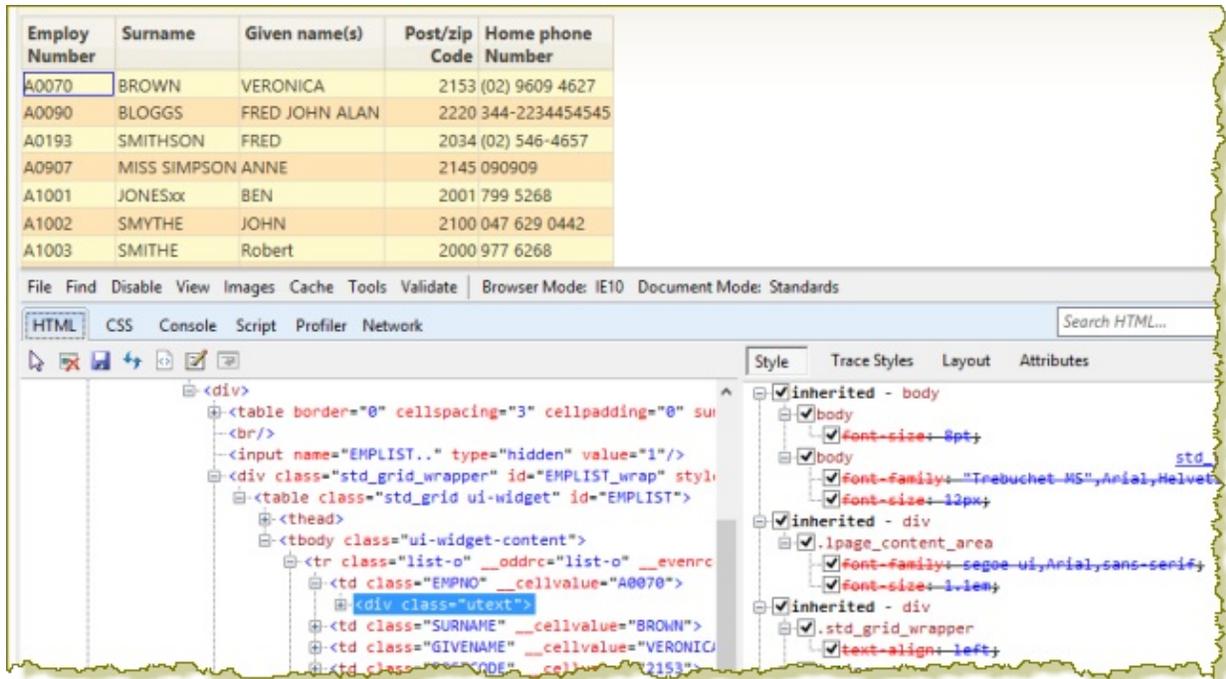
This exercise will demonstrate how to change the appearance of elements in a list. The exercise is about how to identify the elements you wish to change and then implement these changes with your own style sheet.

To use CSS effectively, you must first understand how the how the screen element you are interested in is constructed.

If you are using Internet Explorer 8, 9 or 10 you already have Developer Tools that may be started from the browser Tools menu, or by using F12. Tools like this are essential for understanding how your page is constructed. The *Mozilla Firefox* browser has a similar optional tool known as *Firebug*.



This screen picture shows *Developer Tools* running with a WAM which displays a list of employees:



Note that the *Select element by click*  icon in Developer Tools can be used to select the element on the page that you want to examine.

In this example the table containing the employee list has been selected. The *Developer Tools* then displays the HTML, attributes and styles for the selected element.

Note: *Developer Tools* also has good display source features that will help you to see in more detail how part of the screen is defined. You will use these later in this exercise.

Using *Developer Tools* and selecting the table containing the list, shows that the browselist EMPLST table is defined as: For clarity, some detail has been omitted from this code.

```
<DIV style="WIDTH: 617px" id=EMPLIST_wrap class=std_grid_wrapper>
<TABLE style="CURSOR: default" id=EMPLIST class="std_grid ui-
widget">
<THEAD>
<TR class="list-h ui-widget-header">
<TH class="utext EMPNO std_grid_sort_indicator" __allowsort="true" __moc
Employ
<BR>
Number
<DIV class=std_grid_cell_sizer><!--.--><!------>
```

```

</DIV>
</TH>
.....
<TBODY class=ui-widget-content>
<TR class=list-o __evenrc="list-e" __oddr="list-o">
<TD class=EMPNO __cellValue="A1003">
<INPUT id=EMPLIST.0001.EMPNO class=utext onchange="return isValidTe:
</TD>
<TD class=GIVENAME __cellValue="Robert">
<DIV class=utext>Robert<!--</DIV>
</TD>
.....
</TR>
.....
</TBODY>
</TABLE>

```

Some points to note:

- A list is wrapped by a DIV with the *class* of **std\_grid\_wrapper**.
- The DIV also has an *id* of **<listname>\_wrap**, where **<listname>** is the name of the list in the RDMLX.
- For a grid, the wrapper DIV also has a *class* of **std\_grid\_wrapper**. The *id* is **LANSA\_<gridname>\_wrap** where **<gridname>** is the name assigned to the grid in the Design view.
- The DIV wrapper provides the size and position for the grid, drawing any scrollbars as necessary.
- In a list, the row *class* names are alternated between **list-o** and **list-e**.
- For a grid, the *class* names for odd and even rows are defined by the grid properties **odd\_row\_class** and **even\_row\_class**, which have default values of *odd\_row* and *even\_row*.
- A list's table cells (**<td>** tags) are given a *class* name of the field name. This provides a way to apply styles to specific columns.
- Input fields are given a class that represents the data type of the field. Fields of type alpha, char or string will be "text", "utext" or "ltext" depending on the input case of the field. Fields of type packed, signed, integer, float, date, time or datetime will have a class name of "number". Boolean fields will be "ltext" and all other fields will be "text".

The best way to understand how it all fits together is to look at a few examples.

## Step 1. Create WAM iii Using CSS

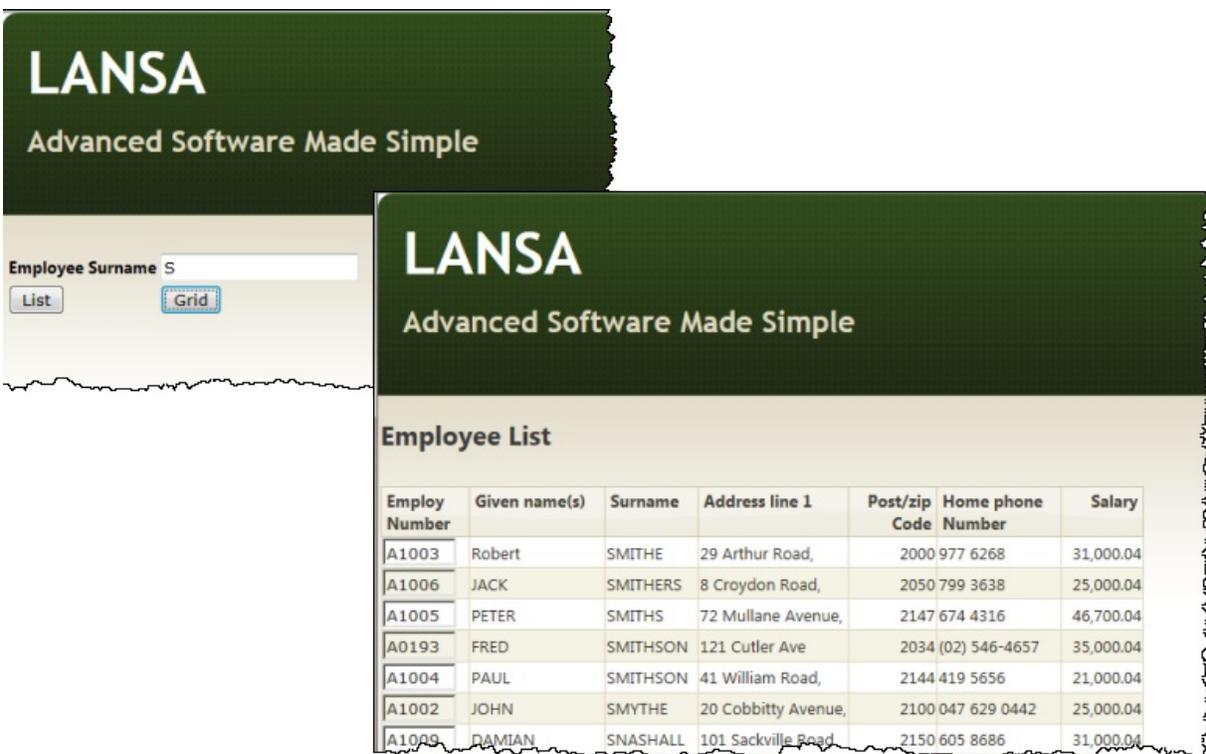
1. Create a new WAM

Name: **iiiUsingCSS**

Description: **Using CSS**

Layout weblet: **iiilay01**

Replace the default RDMLX code with the source provided in [WAM100. Appendix](#). It is a simple WAM that looks like the following. It displays a list of employees as a browse list or a grid.



2. Compile your WAM. Open the **begin** WebRoutine in the design view:

- Extend the table containing Employee Surname by adding a row.
- Add a push button to each new cell and set up the push button properties as follows:

Property	Value
Caption	List
on_click_wrname	Emplist

Caption            **Grid**

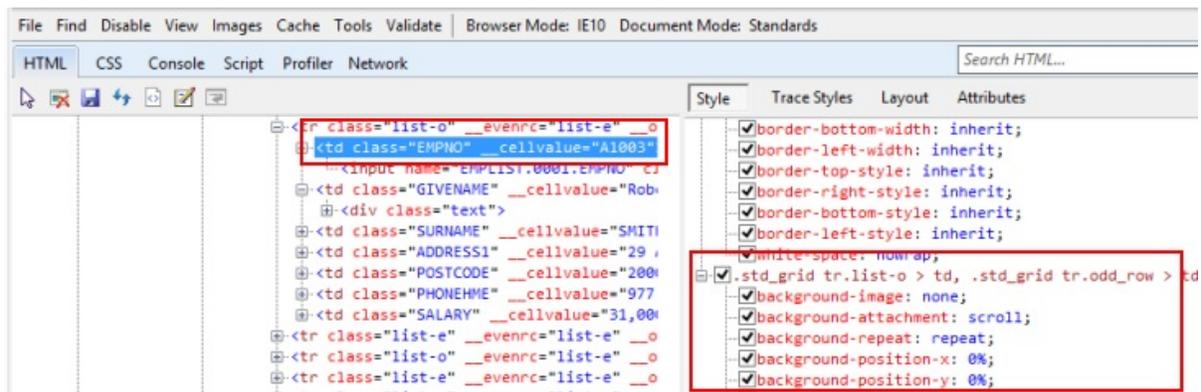
on\_click\_wname **empgrid**

- c. Save your changes.
3. Open the **empgrid** WebRoutine in the *Design* view. Add a *Grid* weblet to the page and link it to list EMPLIST.
4. Save your changes.
5. Run the **begin** WebRoutine in the web browser.
  - a. Enter a partial surname such as "S" or "B"
  - b. Click the *List* push button to display a list of employees.
  - c. Click the *Grid* push button to display a grid of employees.

## Step 2. Create new Style Sheet

In this step you will create a style sheet to control the background color for odd and even rows in a list. To do this you first need to understand how the background color is controlled at the moment.

1. Execute WAM iiiUsingCSS in the browser and run *IE Developer Tools (F12)*.
  - a. Select the "Select" tool by clicking on the  on the toolbar.
  - b. Click anywhere on a list row.
  - c. Using the *HTML* view, select a TD tag, and scroll down the Style view to find the CSS that controls the background.



The CSS selector `.std_grid TR.list-o > TD` applies to all TD tags within an odd row, in a list, which is a table with a class of `std_grid`. The CSS is structured this way, due to the need for cross browser compatibility.

2. In Notepad use *File/New* to start a **new** document and add the following code.

```
.std_grid tr.list-o > td
{
    background-color: #fffacd;
}
.std_grid tr.list-e > td
{
    background-color: #ffe4b5;
}
```

3. Save your style sheet as **iii\_style.css**, where **iii** are your initials. Use *Save as*

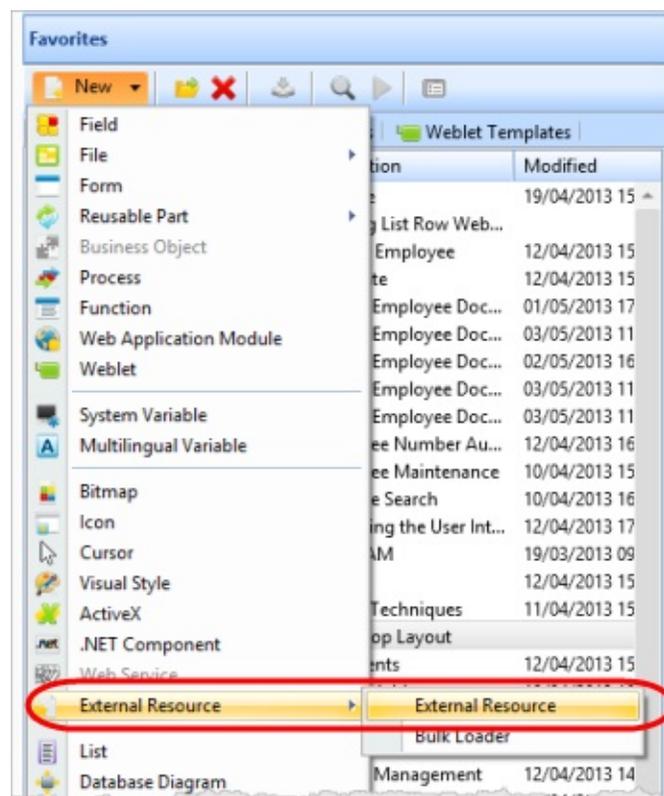
*Type: All Files* to save the file with the css extension.

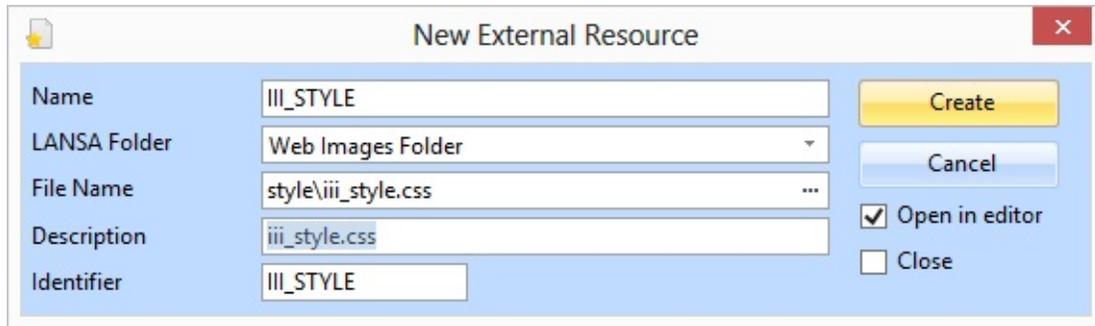
Make sure you save it to: **C:\Program  
Files\LANSA\WebServer\Images\style.**

Leave Notepad open.

### Step 3. Create an External Resource

- In this step you will define an *External Resource* using this style sheet and initially apply this to your WAM layout for *iiiUsingCSS* only.
  - This will mean that style sheet *iii\_style.css* will be applied only to all WebRoutines for the WAM *iiiUsingCSS*.
  - In a later step you will apply the *External Resource* to the common layout **iiilay01**. You will see how it is then applied to all WAMs, sharing this common layout.
  - In a later step you will also change your style sheet so that it targets only lists with an id of *EMPLIST*.
1. On the *Favorites* tab, select *External Resource / External Resource* from the *New* button.





2. Complete the details as shown in the table, but note the following steps:
  - a. Begin by selecting the *File Name* using the *Ellipsis* button and the *Open* dialog to select the new style sheet you saved in the \style folder. The *LANSA Folder* and *Description* will also be automatically completed.
  - c. Uncheck the options to *Open in the Editor*, check the *Close* checkbox.

Name	<b>III_STYLE</b>
LANSA Folder	<b>Web Images Folder</b>
Filename	<b>Style\iii_style.css</b>
Description	<b>iii_style.css</b>

- d. Click *Create* to save your *External Resource* definition.

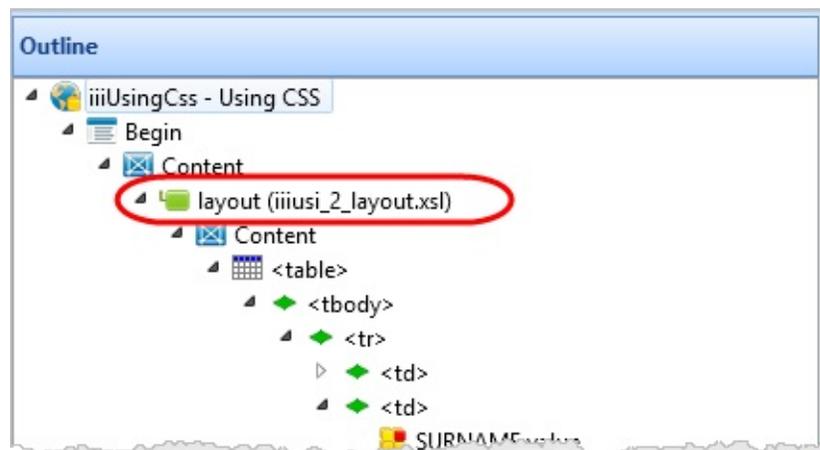
You have created an External Resource entry in the Repository, which can now be used to apply this style sheet to a layout.

## Step 4. Apply Style Sheet to WAM iiiUsingCSS

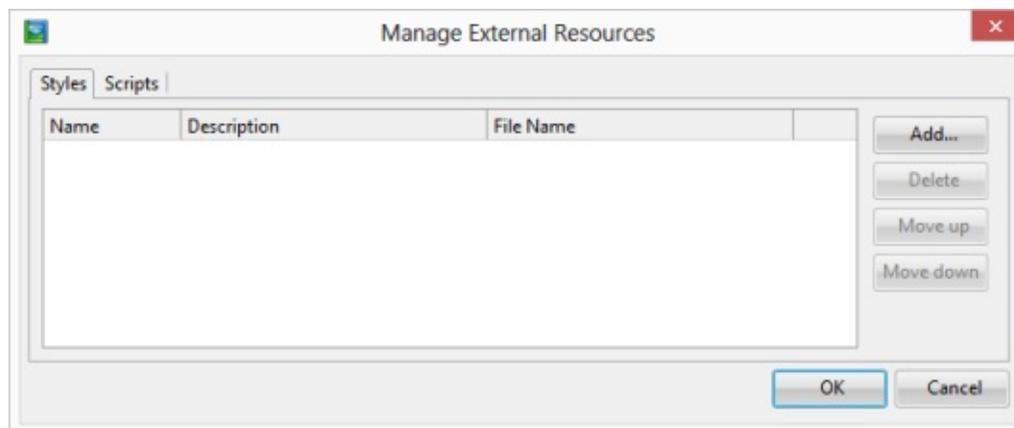
In this step you will open your WAM layout in the editor and add your *External Resource* to this layout. This will apply the style sheet *iii\_style.css* to all WebRoutines in this WAM only.

1. On the *Outline* tab, open your WAM layout in the *Design* view by double clicking on the WAM layout item.

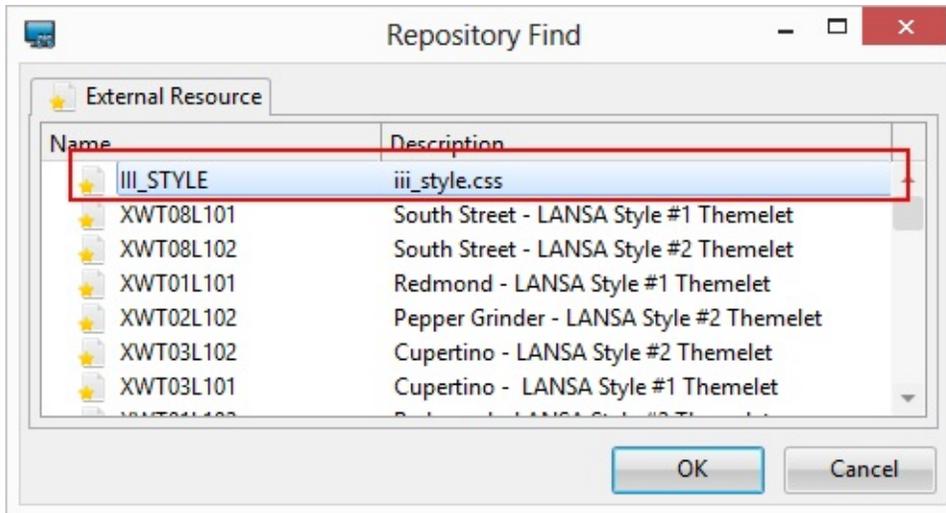
**Note:** The WAM layout is named using the WAM Identifier for example **iiiusi\_2\_layout.xml**. Your name may be slightly different.



2. Select the *Design* ribbon and click the *External Resources* button, to open the *Manage External Resources* dialog



3. Click the Add button, select your external resource and click OK. Click OK to close the *Manage External Resources* dialog.



4. Save the WAM layout.

The editor has added an entry to the WAM layout XSL to apply the style sheet defined in this external resource to the WAM layout.

```
<xsl:import nref="iiilay01.xsl" />
<xsl:import href="std_types.xsl" />
<xsl:output method="xml" omit-xml-declara
  indent="no" />
<wd:external-resources>
  <wd:style name="III_STYLE" />
</wd:external-resources>
<wd:definition>
  <wd:group name="WAM Layouts" />
</wd:definition>
```

5. Close the WAM layout.

6. Execute the WAM **iiiUsingCSS**, WebRoutine **begin** in the browser.

a. Enter a partial surname and display the List page for the employees

### Employee List

Employ Number	Given name(s)	Surname	Address line 1	Post/zip Code	Home phone Number	Salary
A1031	JOHN	BLAKE	3 Woodbury Road	2100	(02) 9668 9235	60,725.00
A0090	FRED JOHN ALAN	BLOGGS	70 MAIN STREET	2220	344-2234454545	20,045.91
A3564	FREDDY	BROWN	121 SMITH STREET	2153	(02) 567-6758	30,000.00
A0070	VERONICA	BROWN	12 Railway Street	2153	(02) 9609 4627	50,125.00

- b. Use the browser back button to return to the begin page, enter a partial surname and display the Grid page:

### Employee Grid

Employ Number	Given name(s)	Surname	Address line 1	Post/zip Code	Home phone Number	Salary
A1031	JOHN	BLAKE	3 Woodbury Road	2100	(02) 9668 9235	60,725.00
A0090	FRED JOHN ALAN	BLOGGS	70 MAIN STREET	2220	344-2234454545	20,045.91
A3564	FREDDY	BROWN	121 SMITH STREET	2153	(02) 567-6758	30,000.00
A0070	VERONICA	BROWN	12 Railway Street	2153	(02) 9609 4627	50,125.00

**Note:** The cascading style sheet has not been applied to the grid. At the moment it defines alternate background colors for a list only.

## Step 5. Apply External Resource to the Common Layout

In this step you will open the common layout **iiilay01** in the editor and apply your External Resource to this layout and test the results.

1. Open the layout **iiilay01** in the editor. You could do this by opening it directly from the *Last Opened* tab on the *Favorites* tab, or by locating it on the *Repository* tab under *Weblets*.
2. With the common layout **iiilay01** open in the editor, as before select the *Design* ribbon and use *External Resources / Manage External Resources* dialog to *Add* the external resource **III\_STYLE** to the layout.
3. Save the common layout and close it.
4. Open the WAM layout for WAM **iiiUsingCSS** and remove the external resource from this layout.
5. Close the WAM layout.
6. Execute the **begin** WebRoutine in the browser for WAM **iiiUsingCSS**. You should obtain the same results as before. Your style sheet for list alternate rows is applied to the list on the List page and is not applied to the grid on the Grid page.
7. Open the WAM **iiiEmpSearch** in the editor and execute the **Search** WebRoutine in the browser. Enter suitable employee numbers to display a list of employees. Note that the new cascading style sheet has been applied to the list.

All WAMs which were defined using the common layout **iiilay01** will have the new cascading style sheet applied. Any web page containing a list will have the CSS applied giving new alternate row background colors.

## Step 6. Make the Style Sheet specific to lists named EMPLIST

As currently defined, the background color changes in your cascading style sheet, apply to all lists.

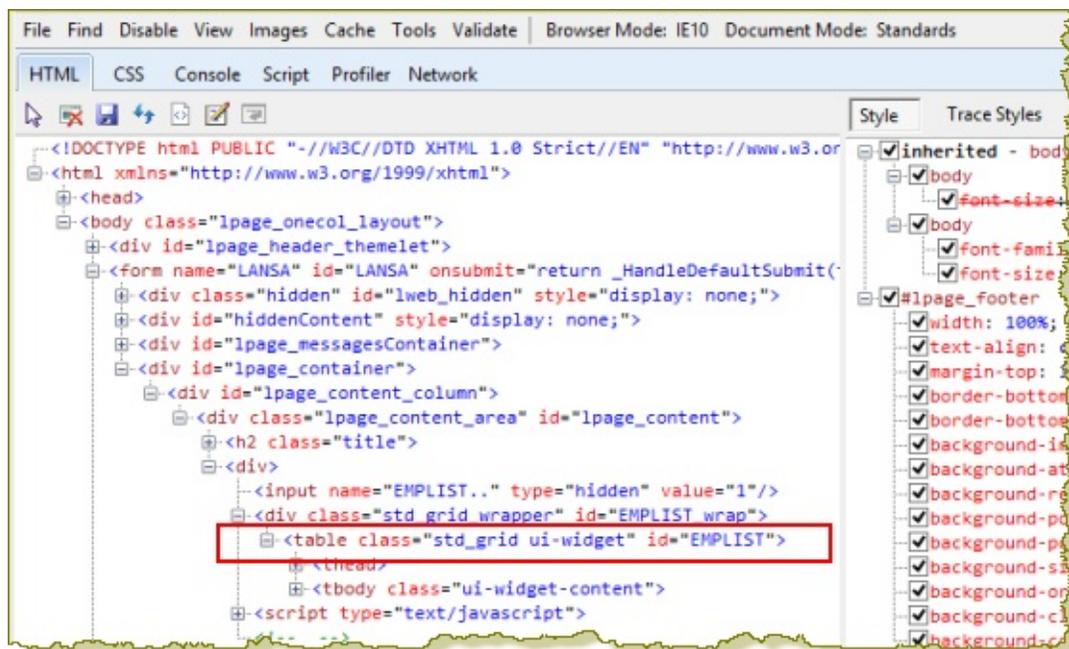
In this step you will make the style sheet specific to lists named EMPLIST.

1. Run the **begin** WebRoutine in the browser for WAM iiiUsingCSS and display a list of employees.

Use the IE Developer tools, as before, to select the List (click on the edge of the List).

Expand the tree on the HTML tab, to show the `<table....>` tag for the list table.

Note this has an *id* of **EMPLIST**. It is given an *id* equal to the list name defined in the RDML.



2. Switch to Notepad, where you should still have your style sheet file open. Change the code to the following and save the file:

```
TABLE#EMPLIST tr.list-o > TD
{
    background-color: #fffacd;
}
TABLE#EMPLIST tr.list-e > TD
```

```
{  
  background-color: #ffe4b5;  
}
```

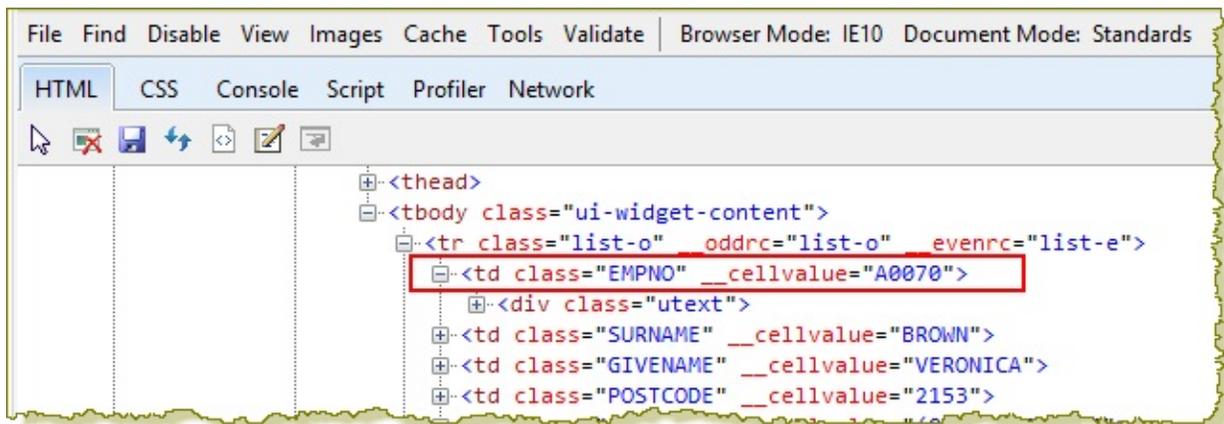
Your styles for odd and even background colors are now defined for lists named EMPLIST only.

11. Run WAM iiiUsingCSS and display the list of employees, which should reflect your style sheet.
12. Run WAM iiiSecMaint to display sections for a department. This list should not reflect your stylesheet, which is now specific to a list with an *id* of **EMPLIST**.

## Step 7. Highlight a Column

In this step, you will investigate how a specific column can be identified and then create style sheet entry to change the background color for this column only.

1. Run the WAM `iiiUsingCSS` to display a list of employees. Using *IE Developer Tools* select the first input field in the first column (Employee Number).



Note that the `<td>` tags for each column have a class equal to the field name. In the first column this is **EMPNO**.

2. Add this code to your style sheet and save the changes:

```
TABLE#EMPLIST tr.list-o > TD.EMPNO
{
  background-color: #d78700;
}
TABLE#EMPLIST tr.list-e > TD.EMPNO
{
  background-color: #d78700;
}
```

This will override the background color for odd and even rows in table cells with a class of **EMPNO**.

3. Run your WAM `iiiUsingCSS` and display a list of employees. Your results should look like the following:

## Employee List

Employ Number	Given name(s)	Surname	Address line 1	Posi C
A1031	JOHN	BLAKE	3 Woodbury Road	
A0090	FRED JOHN ALAN	BLOGGS	70 MAIN STREET	
A3564	FREDDY	BROWN	121 SMITH STREET	
A0070	VERONICA	BROWN	12 Railway Street	

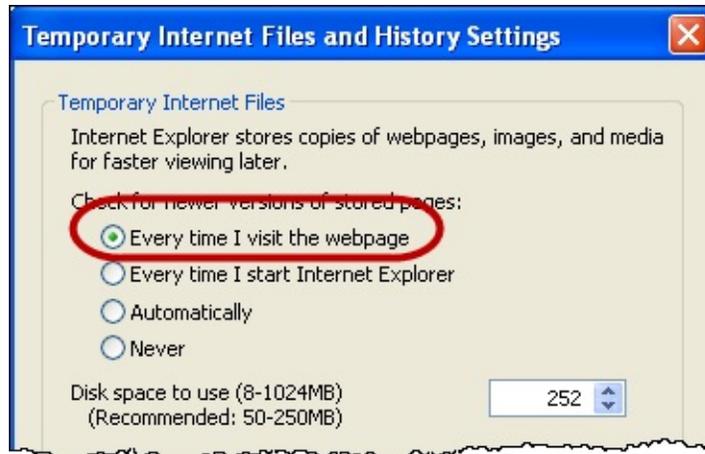
## Summary

### Important Observations

- You should be able to extend this exercise to make other changes to your list's appearance – for example borders, or to other elements on your web pages.
- As long as you are applying a custom CSS over the default, you can't break anything, so feel free to experiment.
- Here you have used a supplied themelet to set the overall appearance of your application. You may want to create your own themelet which replaces the supplied examples.

### Tips & Techniques

- If you have not worked with CSS before, take a look at the tutorials at [www.w3schools.com](http://www.w3schools.com)
- You may also want to take a look at these articles on CSS Selectors: [http://www.456bereastreet.com/archive/200509/css\\_21\\_selectors\\_part\\_1/](http://www.456bereastreet.com/archive/200509/css_21_selectors_part_1/)
- With the exception of the `tr.list-o` and `tr.list-e` styles shown earlier, the default selectors for most grid related styles start with the `.std_grid` class selector. This makes them easier to find in the CSS file and reduces the chances of accidental conflicts with styles used elsewhere (the `tr.list-o` and `tr.list-e` are defined as they are for backwards compatibility reasons).
- In the event of a conflict, the style with the more specific selector will take priority. For example, the default style for grid table cells is defined with `".std_grid tbody td"`. It will override any conflicting properties defined with `".std_grid td"`. So, if a style isn't working as expected, try making it more specific
- When working on web application development ensure that your browser settings check for newer versions of stored pages "Every time I visit the webpage".



- When changing entries in a style sheet, be aware that you may have problems with cached versions. Clear your browser history regularly.

## What You Should Know

- The essential rules for creating and applying style sheets
- How to use the IE Developer Tool to understand the construction of your web pages at a detailed level.

## WAM100. Appendix

Use the following RDMLX source code to create iiiUsingCSS in Step 1 of this exercise.

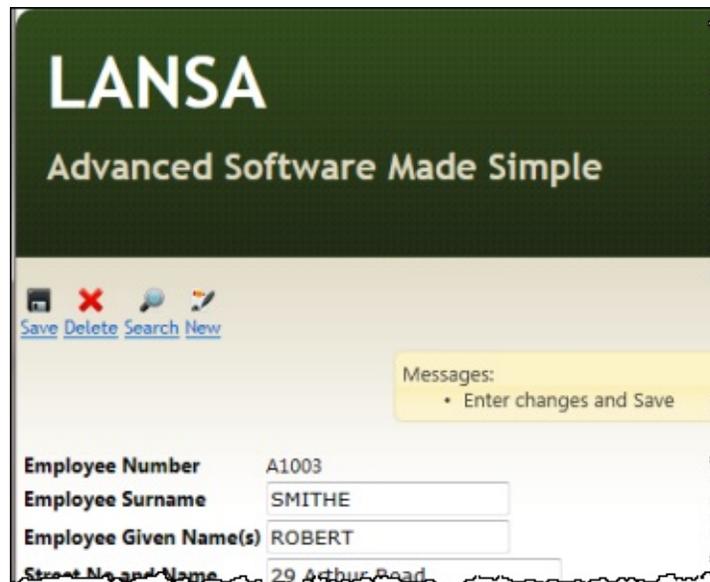
```
Def_List Name(#emplist) Fields(#empno (#givenname *out) (#surname *out) (#
Define Field(#hidesave) Type(*char) Length(1)
Define Field(#hidedel) Type(*char) Length(1)
Define Field(#hidenew) Type(*char) Length(1)
Define Field(#hidesrch) Type(*char) Length(1)
Define Field(#empnow) Reffld(#empno)
Web_Map For(*both) Fields((#stdrentry *hidden) (#empnow *hidden))
WebRoutine Name(Begin)
Web_Map For(*output) Fields(#surname)
#hidedel #hidesave #hidesrch := Y
Endroutine
WebRoutine Name(empgrid) Desc('Employee Grid')
Web_Map For(*input) Fields(#surname)
Web_Map For(*output) Fields((#emplist *private))
#hidedel #hidesave := Y
Execute Subroutine(bldlist)
Endroutine
WebRoutine Name(emplist) Desc('Employee List')
Web_Map For(*input) Fields(#surname)
Web_Map For(*output) Fields(#emplist)
#hidedel #hidesave := Y
Execute Subroutine(bldlist)
Endroutine
Subroutine Name(bldlist)
Clr_List Named(#emplist)
Select Fields(#emplist) From_File(pslmst2) With_Key(#surname) Generic(*ye
Add_Entry To_List(#emplist)
Endselect
Endroutine
```

## WAM105 - Create Your Own Weblet

### Objectives

To create Toolbar Menu Item weblet and a Toolbar weblet. A simple WAM application will then be used to test the Toolbar weblet.

The finished application will look like the following:



In order to complete this exercise, you must complete the following:

Review [Weblets](#)

Step 1. [Create Toolbar Menu Item Weblet](#)

Step 2. [Create Toolbar Weblet](#)

Step 3. [Complete Definition of Toolbar Menu Item Weblet](#)

Step 4. [Setup iii\\_toolbar\\_menuitem Properties in iii\\_toolbar](#)

Step 5. [Apply Toolbar Weblet to an Employee Maintenance WAM](#)

[Summary](#)

### Before You Begin

This exercise does not depend on knowledge gained from all the preceding exercises. It is recommended that the following have been completed:

[WAM005 - Create Your First WAM](#)

[WAM010 - Using WEB\\_MAPs](#)

[WAM015 - Working Lists](#)

## WAM020 - WAM Navigation

### Weblets

Layouts are a type of Weblet. They allow you to customize the overall look and feel of a web site or web application.

Weblets are also building blocks of your WAM HTML page. They can be categorized into 2 main groups.

- Primitive Weblets typically have a 1-to-1 relationship with an HTML Tag/Element (eg: Push Button, Checkbox, Combo Box, Anchor, Clickable Image, an Input box, etc)
- Composite Weblets provide additional functionality combining Javascript, CSS, Primitive Weblets and HTML (eg: Grid, Tabsheets, etc)

### XSL Templates

The `<xsl:template name="my_template_name"> </xsl:template>` element is very similar to the SUBROUTINE command used to define Subroutines in LANSAs.

Parameters can be received by a template by defining

`<xsl:param name="param_name" />` elements within the template.

### Calling XSL Templates

The `<xsl:call-template name="layout-form.private">` element is very similar to the EXECUTE command used to call subroutines in LANSAs.

Parameters can be passed on the call using this element:

`<xsl:with-param name="param_name" select="param_value"/>`

To learn more about XSLT, see <http://www.w3schools.com/xsl/default.asp>

## Step 1. Create Toolbar Menu Item Weblet

In this step you will create a Toolbar Menu Item from which the toolbar will be built. If you have some basic HTML knowledge, then you will know that a menu item is essentially an `<a>` anchor tag that can have an href, image and alternate text elements etc associated with it. You should also know that in a WAM application an "href" will usually call a JavaScript function to call a WAM / WebRoutine.

1. Create a new weblet. From the *File* menu select *New / Weblet* in the Visual LANSa Editor.
  - a. In Name and Description, replace iii with your initials.



The screenshot shows a dialog box titled "New Weblet" with a close button in the top right corner. The dialog contains the following fields and controls:

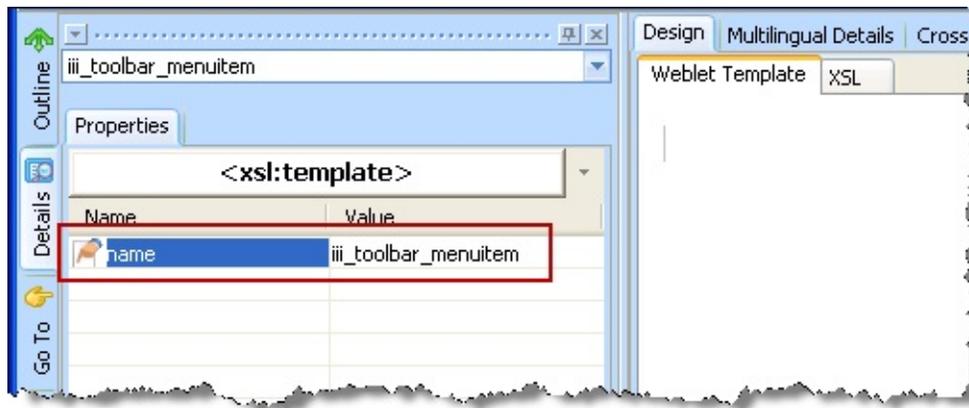
- Name:** A text input field containing "iii\_toolbar\_menuitem".
- Description:** A text input field containing "Toolbar Menu Item".
- Weblet Group:** A dropdown menu with "Custom Weblets" selected.
- Layout Weblet:** A checkbox that is currently unchecked.
- Buttons:** A yellow "Create" button and a blue "Cancel" button are located on the right side of the dialog.

- b. Select **Custom Weblets** as the *Weblet Group*.
- c. Press *Create* to create your weblet and the **Custom Weblets** group. Type in a group name of **Custom Weblets** if this does not exist. The weblet will open in the *Design* view.

The XSL Source for your weblet should currently look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) 2003, 2011 LANSA -->
<!-- XHTML weblet -->
<!-- $Revision:: 3 $ -->
<xsl:transform version="1.0" exclude-result-prefixes="lxml wd tsml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
  xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-Metadata"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:import href="std_types.xsl" />
  <xsl:output method="xml" omit-xml-declaration="yes" encoding="UTF-8"
    indent="no" />
  <wd:external-resources />
  <wd:definition>
    <wd:use-design-layout />
    <wd:group name="Custom Weblets" />
  </wd:definition>
  <xsl:template name="unnamed">
    <!-- Give your template appropriate name and type in your XSL here -->
  </xsl:template>
</xsl:transform>
```

2. Select the *Weblet Template* tab and use the *Details* tab to enter a *name* of **iii\_toolbar\_menuitem**. Replace **iii** with your initials.



3. Save your changes.
4. You will now add the skeleton HTML code for the anchor tag, inside the

**<xsl:template ...>...</xsl:template>**, as follows:

Copy the following code `<a href="jav..... code` and paste it immediately following the lines:

Code should be copied from *WAM Tutorials* in the Visual LANSAs online guide.

The new code is highlighted in red.

```
<xsl:template name="iii_toolbar_menuitem">
  <!-- Give your template an appropriate name and type in your XSL here -->
  <a href="javascript:">    
      Menu Text
    </span>
  </a>
```

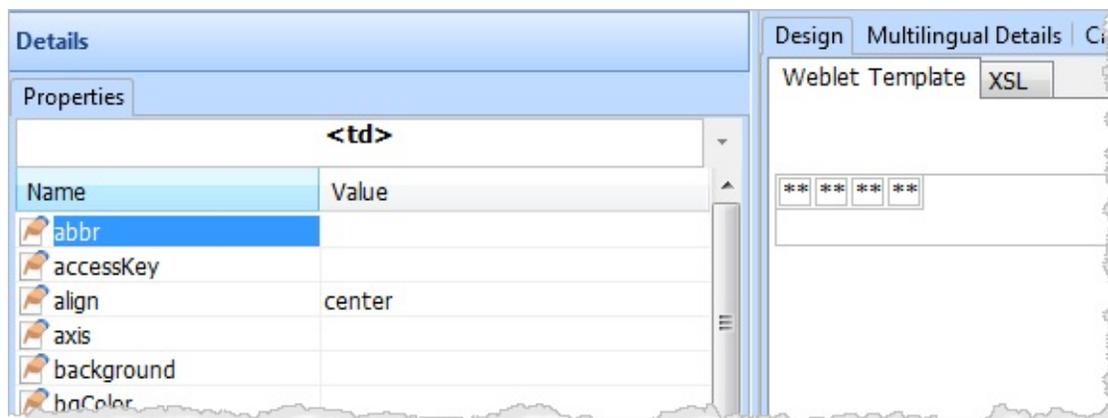
In later steps you will complete this outline for the anchor tag code.

4. Click the Save  button on the editor *Toolbar* to save your changes.

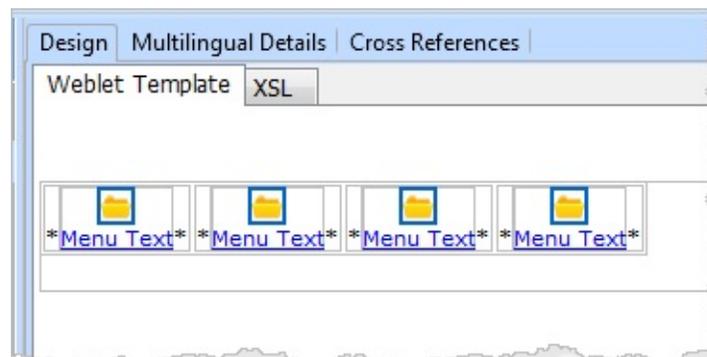
## Step 2. Create Toolbar Weblet

In this step you will create a second weblet to build your Toolbar weblet. This will simply consist of a single row table with 4 cells.

1. As before, create a new weblet using *New / Weblet*, *Name* = **iii\_toolbar**, *Description* = **Toolbar**, *Group* = **Custom Weblets**. Replace **iii** with your initials.
2. On the *Details* tab change the weblet *name* to **iii\_toolbar**.
3. In the *Design* view use the context menu to *Insert HTML / Table* with 1 row and 4 columns.
4. Save your changes.
5. Select each cell in the toolbar table and use the *Details* tab to set its *align* property to **center**.



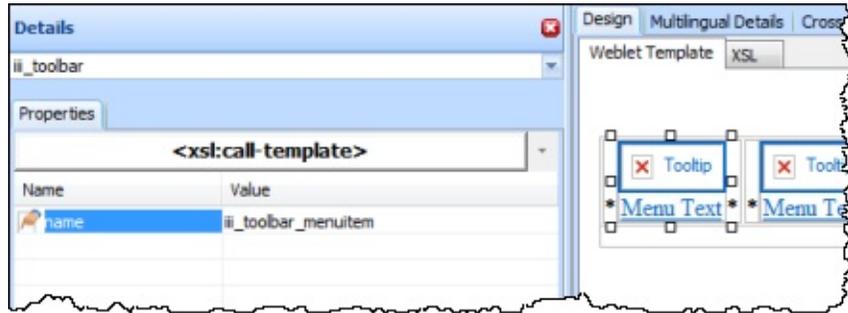
6. Switch to the *Favorites* tab, find your *iii\_toolbar\_menuitem* weblet. Drag and drop a *iii\_toolbar\_menuitem* into each of the four cells in the toolbar. Your toolbar should now look like the following in the Weblet Template design view.



7. Use the cursor keys to position into each cell (<td> tag) and delete the \* place holder characters.

Save your changes.

8. Select one of the toolbar menu items and select the *Details* tab. Notice that at present, the menu items have no properties that can be set.



### Step 3. Complete Definition of Toolbar Menu Item Weblet

In this step you will complete the coding of the toolbar menu item weblet. You will add code to:

- Define weblet parameters
  - Define elements of the anchor tag (for example, IMG ALT) as XSL variables
  - Complete JavaScript for HREF to set reentry field value and call wam / WebRoutine
  - Condition the <A> HREF and IMG tags based on a \$hide\_if variable
  - Add weblet parameter tooltips.
1. Switch to the iii\_toolbar\_menuitem or open it in the editor if necessary. Select the XSL tab.
  2. Immediately following the **<xsl:template name="iii\_toolbar\_menuitem">** tag, paste in the following code to define the weblet parameters. Review the comments for each parameter to see where it will be used.

```
<!-- Used to set the Menu Text on the toolbar image -->
  <xsl:param name="menu_text" wd:type="std:mtxt_variable" select="Capt
  <!-- Used to set the image use for the toolbar Icon -->
  <xsl:param name="menu_image" wd:type="std:html_img_relative"
    select="/icons/normal/16/folder_16.png" />
  <!-- Used to set the ALT tag on the toolbar IMG tag -->
  <xsl:param name="tooltip_text" wd:type="std:mtxt_variable"
    select="Caption" />
  <!-- Used to set the Rentry Field Name when the toolbar Icon is clicked --
  >
  <xsl:param name="reentryfield"
    wd:type="std:field_name_in[wam=$on_click_wamname]
[wabrtn=$on_click_wrname]"
    select="STDREENTRY" wd:tip_id="" />
  <!-- Used to set the Rentry Field Value when the toolbar Icon is clicked-->
  <xsl:param name="reentryvalue" select="M" wd:tip_id="" />
  <!-- Used to set the Menu Text on the toolbar image -->
  <xsl:param name="hide_if" wd:type="std:boolean" select="false()"
    wd:tip_id="" />
```

```

<!--
- Used to specify the WAMNAME to call when toolbar Icon is clicked -->
  <!-- It will default to the current WAM if no value is specified -->
  <xsl:param name="on_click_wamname" wd:type="std:wam"
    select="/lxml:data/lxml:context/lxml:webapplication" wd:tip="" />
  <!-- Used to specify the WebRoutine to call when toolbar Icon is clicked --
>
  <xsl:param name="on_click_wrname"
    wd:type="std:webroutine[wam=$on_click_wamname]" wd:tip="" />

```

This block of code defines the *parameters* that can be passed into the template when the toolbar menu item template is called, in a similar way to calling a subroutine with parameters. Once defined these become the *properties* that can be set in the *Design* view for a web page that uses this weblet.

For example: a parameter, named *menu\_text*, has a default value of 'Caption'. In the completed anchor tag code following, note that the variable **\$menu\_text** is used as the caption text below the toolbar item image.

3. Save your changes.

4. You will now complete the code for the <A> anchor tag. Copy the following code and paste it to **replace** the skeleton code for the <A> tag, which you placed there earlier.

```

<a href="javascript:InsertHidden(document.LANSA,'{$reentryfield}','{$reentr
  
  <br />
  <span class="std_menuitem">
    <xsl:value-of select="$menu_text" />
  </span>
</a>

```

Note the following points about these changes:

The value for the alt tag has been replaced with a variable \$tooltip\_text

The value for image file name in the src tag has been replaced with a variable menu\_image

The menu text inside the span tag is now generated by an <xsl:value-of which puts the value of variable \$menu\_text

The href for the A tag has been defined with JavaScript code that passes \$entryfield and \$reentryvalue variables to the InsertHidden function

The href code also runs the HandleEvent function passing \$on\_click\_wamname and \$on\_click\_wrname variables.

5. Save your changes
6. In this step you will add xsl code to condition the anchor tag, based on the **\$hide\_if** parameter.

**Note:** The `<xsl:if>` element must have an `</xsl:if>` end tag. The `<xsl:if>` must surround the entire `<A HREF . . . .>` tag. Add the highlighted code only:

```
<xsl:if test="not($hide_if)">
  <a href="javascript: . . . .
  </a>
</xsl:if>
```

**Hint:** The XSL editor autocomplete function will generate the `</xsl:if>` when you complete the beginning tag, `<xsl:if . . . >`. Move this to the required position after the `</a>` tag.

The variable **\$hide\_if** is a Boolean, with a default value of 'false' as shown in the parameter definitions.

7. In this step you will add Weblet Parameter tooltips by copying in the following code, following the `</wd:definition>`. Note this is the end tag for the block beginning `<wd:definition>`.

Replace **iii** in `<wd:template name` with your initials.

```
<wd:template name="iii_toolbar_menuitem">
  <wd:description icon="icons/userdefn.ico">
    <wd:name lang="ENG">iii Toolbar Menu Item</wd:name>
  </wd:description>
  <wd:param name="menu_text">
    <wd:tip lang="ENG">Menu Text to display below the image on the toolt
  </wd:param>
  <wd:param name="menu_image">
    <wd:tip lang="ENG">Image to display on the toolbar menu item</wd:tip
  </wd:param>
  <wd:param name="tooltip_text">
    <wd:tip lang="ENG">Tooltip text to display on the toolbar menu item</\
```

```
</wd:param>
</wd:template>
```

**Note:**

- Tags such as `<wd:template name="iii_toolbar_menu_item">` have a namespace of **wd**. They are LANSA defined weblet design tags, defined by the standard that is referenced at the top of the weblet XSL. See: **`xmlns:wd=http://www.lansa.com/2002/XSL/Weblet-Design`**.
- Standard XSL tags have a namespace of **xsl**, for example, `<xsl:if.....>`
- The `<wd:template . . . . .>` code is used by the Design view to define the icon and description used in the list of weblets, and the tooltip text for the weblet parameters.

8. Look towards the top of your toolbar XSL to find the statement:

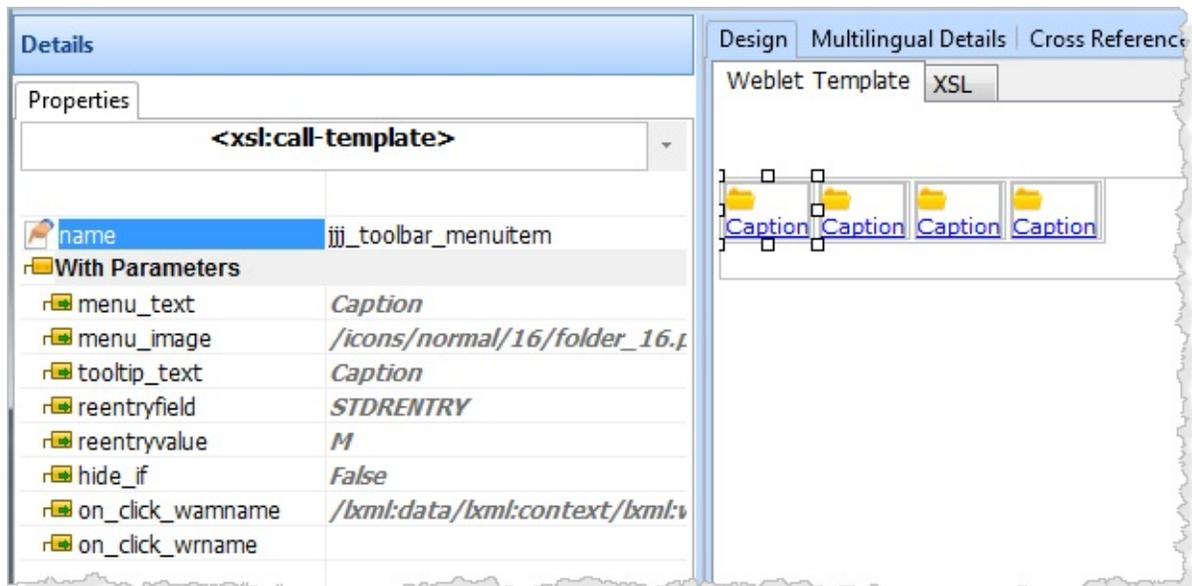
```
<xsl:import href="std_types.xsl" /> and add the following line after that line:
<xsl:import href="std_keys.xsl" />
```

The **std\_keys** XSL defines **xsl:key**'s such as "field-caption" and "field-value" that are used during transformation to extract data from the Data XML output via the WebRoutine.

## Step 4. Setup iii\_toolbar\_menuitem Properties in iii\_toolbar

In this step you will complete the definition of the toolbar by setting up the properties for each of the four menu items.

1. If necessary, open your **iii\_toolbar** in the editor.
2. In the Design view select the first menu item. The Details tab should now contain properties that can be set for each menu item.



3. Click on each of the 4 **Menu items** in order and set their **weblet properties** on the **Details** tab as follows:

On the *Details Tab*, make sure to take note of the tooltip/help text provided for each weblet property. Some of these tooltips are shipped with LANSAs for standard weblet parameter types like WAMName and WebRoutine name but other custom properties like Menu Text also have tooltip text that was added to the weblet definition.

<i>First Menu Item</i>	
<i>Property Name</i>	<i>Value</i>
menu_text	Save

menu_image	icons/normal/16/diskette_16.png
tooltip_text	Save a changed or new employee
reentryfield	STDREENTRY (the default value)
reentryvalue	S
hide_if	#HIDESAVE='Y'
on_click_wamname	Leave as default (current WAM)
on_click_wrname	#WRNAME

<i>Second Menu Item</i>	
<i>Property Name</i>	<i>Value</i>
menu_text	Delete
menu_image	icons/normal/16/cross_16.png
tooltip_text	Deletes the current employee
reentryfield	STDREENTRY (the default value)
reentryvalue	D
Hide_if	#HIDEDEL='Y'
on_click_wamname	Leave as default (current WAM)
on_click_wrname	Maint

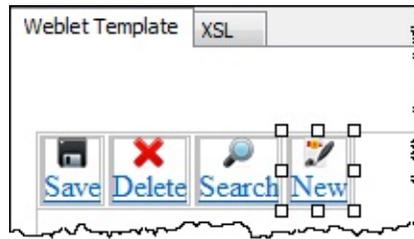
<i>Third Menu Item</i>	

<i>Property Name</i>	<i>Value</i>
menu_text	Search
menu_image	icons/normal/16/zoom_16.png
tooltip_text	Switch to the Search web page
reentryfield	STDREENTRY (the default value)
reentryvalue	M
Hide_if	#HIDESRCH='Y'
on_click_wamname	Leave as default (current WAM)
on_click_wrname	Begin

<i>Fourth Menu Item</i>	
<i>Property Name</i>	<i>Value</i>
menu_text	New
menu_image	icons/normal/16/contract_16.png
tooltip_text	Switch to the create new employee web page
reentryfield	STDREENTRY (the default value)
reentryvalue	N
hide_if	#HIDENEW='Y'
on_click_wamname	Leave as default (current WAM)
on_click_wrname	New

Use the Ellipsis button to find an image for the menu\_image value.

4. Save your changes. You have now completed the definition of your toolbar weblet, which should look like the following:



## Step 5. Apply Toolbar Weblet to an Employee Maintenance WAM

In this step you will create a new WAM based on supplied RDMLX code, set up the web page for each WebRoutine and then add the toolbar to the WAM layout.

1. Create a new WAM **iiiEmpMaint\_TB – Employee Maintenance with Toolbar**, using Weblet Template **iiilay01**, based on the code supplied in [WAM105. Appendix A A](#).

2. Compile the WAM.

3. Examine the WAM code and note the following:

Fields HIDEDEL, HIDESRCH, HIDENEW, HIDESAVE are globally mapped as `iden` fields – these fields control whether the tool bar icons are shown.

Field WRNAME is also mapped globally as a hidden field. This is used to control which WebRoutine the Save button calls.

The WebRoutines set the value of the "HIDExxx" fields as appropriate. For example the Begin WebRoutine shows only the New toolbar icon.

4. Open the **begin** WebRoutine in the *Design* view, add a row to the table and add pushbuttons with a *caption* of **List** and **Grid** and with *on\_click\_wrname* values of **emplist** and **empgrid** respectively. These buttons do not require a *submitFieldValues* property.



5. Open the **emplist** WebRoutine in the *Design* view. Drop an *Anchor* weblet onto the employee number field in the first list column. Set up the *Anchor* weblet properties as shown:

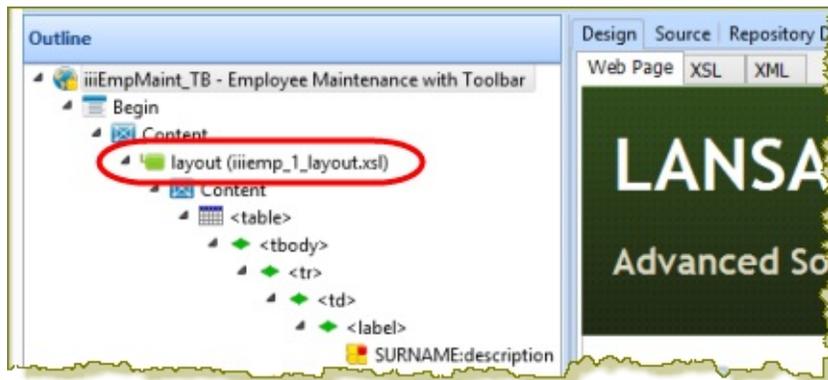
Property	Value
currentrowhfield	<b>EMPNO</b>
currentrownumval	<b>\$EMPNO</b>
Reentryvalue	<b>M</b>
on_click_wrname	<b>maint</b>

Your list should now look like the following:

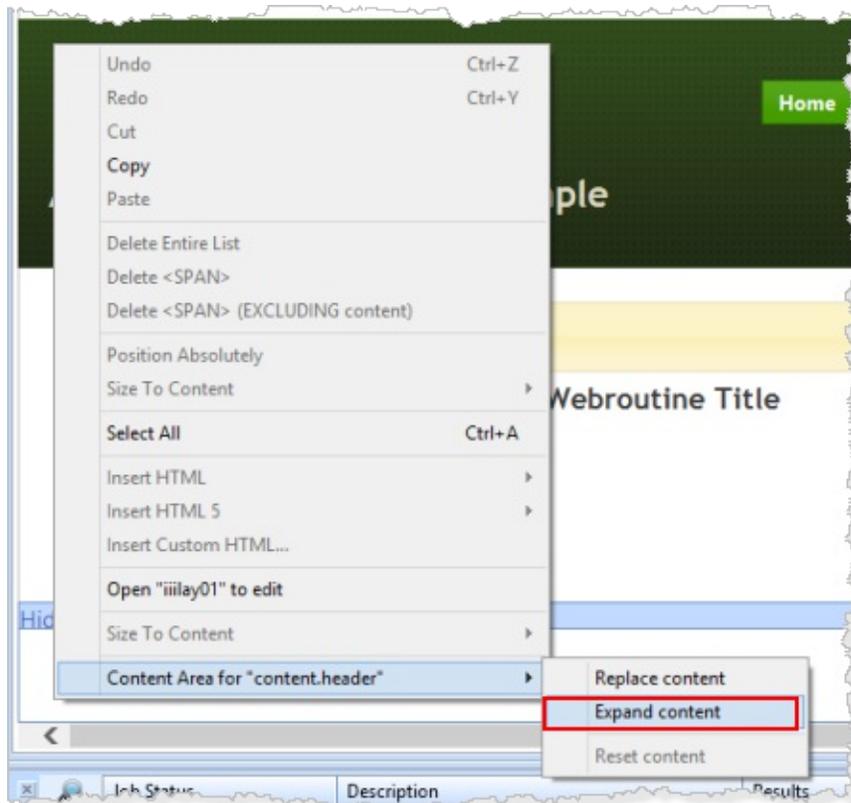
Employ Number	Given name(s)	Surname	Dept Code	Section Code	Salary
Value EMPNO 1	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	Value DEPARTMENT 1	Value SECTION 1	123,456,789.12
Value EMPNO 2	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	Value DEPARTMENT 2	Value SECTION 2	123,456,789.12
Value EMPNO 3	ABCDEFGHIJKLMNQRST	ABCDEFGHIJKLMNQRST	Value DEPARTMENT 3	Value SECTION 3	123,456,789.12

6. Save your changes.
7. Select the  *Outline* tab. Double click on the layout **iiiempma\_layout.xml** to open it.

**Note:** The layout name uses the WAM **Identifier**, which may have a different value in your Visual LANSA.



8. With the wam layout open in the editor, click on the header area and using the context menu, select *Content Area for "content.header"* and *Expand content*.



The layout will be redisplayed, with the change area highlighted in red.



9. Select the Add Content area. On the *Weblet Templates* tab, select *Custom Weblets* and drop your **iii\_toolbar** weblet onto the layout, inside red **Add content** area. This is actually a span tag, with inline styles set, giving a red background and yellow text. Your layout will now look like the following:



10. Click on the Add content area to select it, and use the *Details* tab to clear its inline styles. This will remove the red background, yellow color and black borders. It also has a margin of 10px which could be retained, if required.

Delete the \*\* Add content . . . . text from inside the span, leaving just your toolbar weblet.

Your page should look like the following:



11. Save and close the wam layout. You have added the toolbar weblet into this WAM layout only. If it was required in all your WAMs you could have added it into your common layout, **iiilay01**.

12. Execute your WAM in the browser by running the **Begin** WebRoutine. Test the operation of the toolbar. The toolbar items will be hidden when not required. For example, the **Begin** WebRoutine will display only the *New* toolbar item.

## Summary

### Important Observations

- Once you understand the elements of a weblet's code, simple weblets can be developed quite easily.
- More complex weblets will require good XSL, HTML and possibly Javascript knowledge in order to design and create them.

### Tips and Techniques

- Develop a weblet in small steps with frequent testing.
- It is often useful to add text between tags that will highlight areas in your prototype design.

### What I should now know

- XSL tags with the wd namespace (for example, <wd:template name= . . . >) are LANSA weblet design tags which are used by the *Design* view.

## WAM105. Appendix A

Use the following RDMLX source code to create iiiEmpMaint\_TB in Step 5 of this exercise.

```
DEF_LIST NAME(#emps) FIELDS(#empno (#givename *out) (#surname *ou
DEFINE FIELD(#hidesave) TYPE(*char) LENGTH(1)
DEFINE FIELD(#hidedel) TYPE(*char) LENGTH(1)
DEFINE FIELD(#hidenew) TYPE(*char) LENGTH(1)
DEFINE FIELD(#hidesrch) TYPE(*char) LENGTH(1)
DEFINE FIELD(#empnow) REFFLD(#empno)
Define Field(#wrname) Type(*char) Length(50)
GROUP_BY NAME(#empmnt) FIELDS(#SURNAME #GIVENAME #ADDI
GROUP_BY NAME(#empadd) FIELDS(#EMPNO #SURNAME #GIVENAM
WEB_MAP FOR(*output) FIELDS((#hidesave *hidden) (#hidedel *hidden) (#
WEB_MAP FOR(*both) FIELDS((#stdrentry *hidden) (#empnow *hidden))
WebRoutine NAME(Begin)
WEB_MAP FOR(*output) FIELDS(#surname)
#hidedel #hidesave #hidesrch := Y
ENDROUTINE
WebRoutine NAME(empgrid)
WEB_MAP FOR(*input) FIELDS(#surname)
WEB_MAP FOR(*output) FIELDS((#emps *private))
#hidedel #hidesave := Y
EXECUTE SUBROUTINE(bldlist)
ENDROUTINE
WebRoutine NAME(emplist) DESC('Employee List')
WEB_MAP FOR(*input) FIELDS(#surname)
WEB_MAP FOR(*output) FIELDS(#emps)
#hidedel #hidesave := Y
EXECUTE SUBROUTINE(bldlist)
ENDROUTINE
WebRoutine NAME(maint)
WEB_MAP FOR(*both) FIELDS((#empno *out) #empmnt)
#hidedel #hidesave #hidesrch #hidenew := N
#wrname := 'maint'
CASE (#stdrentry)
WHEN (= S)
```

```

UPDATE FIELDS(#empmnt) IN_FILE(pslmst) WITH_KEY(#empnow) VAL_
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('Employee changed')
TRANSFER TOROUTINE(begin)
ENDIF
#EMPNO := #EMPNOW
WHEN (= D)
DELETE FROM_FILE(pslmst) WITH_KEY(#empnow) VAL_ERROR(*next)
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('Employee deleted')
TRANSFER TOROUTINE(begin)
ENDIF
#EMPNO := #EMPNOW
OTHERWISE
FETCH FIELDS(#empmnt) FROM_FILE(pslmst) WITH_KEY(#empno)
#empnow := #empno
MESSAGE MSGTXT('Enter changes and Save')
ENDCASE
ENDROUTINE
WebRoutine NAME(new)
WEB_MAP FOR(*both) FIELDS(#empadd)
#hidedel #hidenew := Y
CASE (#stdrentry)
WHEN (= N)
#empadd := *default
MESSAGE MSGTXT('Enter details and Save')
WHEN (= S)
INSERT FIELDS(#empadd) TO_FILE(pslmst) VAL_ERROR(*next)
IF_STATUS IS(*OKAY)
MESSAGE MSGTXT('New Employee added')
ENDIF
ENDCASE
#wrname := 'new'
ENDROUTINE
SUBROUTINE NAME(bldlist)
CLR_LIST NAMED(#emps)
SELECT FIELDS(#emps) FROM_FILE(pslmst2) WITH_KEY(#surname) GE
ADD_ENTRY TO_LIST(#emps)
IF (#listcount = 15)

```

```
MESSAGE MSGTXT('First 15 entries shown only')  
LEAVE  
ENDIF  
ENDSELECT  
ENDROUTINE
```

## WAM105. Appendix B

The Utility weblet **iii\_keys** may be created by copying the following XSL code, into a new weblet definition, to replace the default code.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) 2002 LANSA -->
<!-- LANSA Runtime-Data XML Webroutine XSLT keys -->
<!-- $Workfile:: std_keys.xsl $ -->
<!-- $UTCDate:: 2011-02-17 23:35:34Z $ -->
<!-- $Revision:: 7 $ -->
<xsl:transform version="1.0" exclude-result-prefixes="lxml wd"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:tsml="http://www.lansa.com/2002/XML/Generation-Metadata"
xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
xmlns="http://www.w3.org/1999/xhtml">
<xsl:key name="field-caption"
match="/lxml:data/lxml:fields/lxml:field/lxml:caption"
use="../@name" />
<xsl:key name="field-value"
match="/lxml:data/lxml:fields/lxml:field/lxml:value" use="../@name" />
<xsl:key name="option" match="/lxml:data/lxml:options/lxml:option"
use="@name" />
<xsl:key name="variable" match="/lxml:data/lxml:variables/lxml:variable"
use="@name" />
<xsl:key name="weblet" match="/lxml:data/lxml:weblets/lxml:weblet"
use="@name" />
<xsl:key name="list" match="/lxml:data/lxml:lists/lxml:list" use="@name" />
<xsl:key name="jsonlist" match="/lxml:data/lxml:lists/lxml:json-list"
use="@name" />
<xsl:key name="tsmllist"
match="lxml:data/tsml:data[@used_by =
'LANSA_XHTML']/tsml:lists/tsml:list"
use="@name" />
<xsl:key name="tsmlcolumn"
match="lxml:data/tsml:data[@used_by =
'LANSA_XHTML']/tsml:lists/tsml:list/tsml:list-
entries/tsml:entry/tsml:column"
```

```
use="concat(ancestor::tsml:list/@name,'_',@name)" />  
<wd:definition>  
<wd:group name="Utility Weblets" />  
</wd:definition>  
</xsl:transform>
```

## WAM110 - Create Your Own Layout Weblet

### Objectives

As you have already seen, the Web Application Layout Manager Wizard enables you to create your own layout based on one of the supplied designs and themes. Bear in mind that the appearance of this style of layout could be considerably modified by simply changing the CSS associated with it. This standard layout can then be applied to each WAM you create.

You may require your WAM layouts to closely resemble your company web site standards and appearance. If you are building a business to consumer application, then this will certainly be the case. This exercise demonstrates how you can start from your own layout and embed this within a layout weblet so that it can be applied to your WAM application.

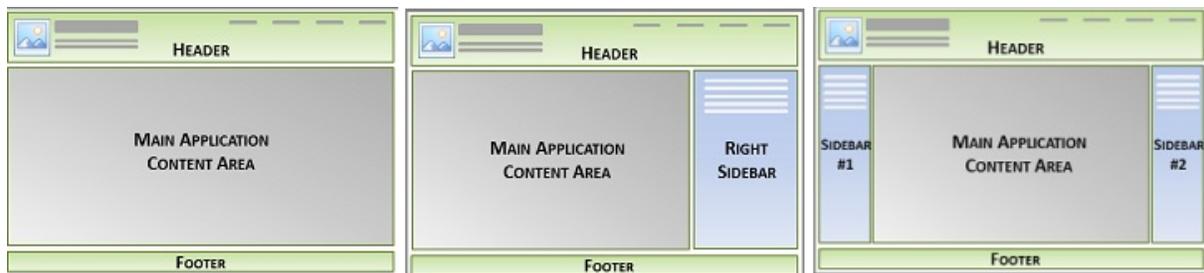
### What is a WAM Layout?

A WAM layout is a specific type of weblet that is used to give structure to the web page associated with a webroutine and to interface with any documents referenced in the layout definition for functional or aesthetic values.

By default, each WAM has an associated WAM layout weblet, which is used as the basis for any presentation associated with the WAM's webroutines. A single WAM layout is generated for each WAM regardless of how many webroutines are defined within the WAM. If your web application includes multiple WAMs, the same layout can be applied to all the WAMs in your application. This way, you can guarantee a consistent interface.

As a visual element, a WAM layout typically provides the structure for any resulting web page. In this role, a WAM layout can define any titles, menus, message presentation or logos to be displayed. The WAM layout also controls the Cascading Style Sheet to be applied.

Your layout could have literally any appearance, but with transactional systems there will always be some kind of main content area, for example:



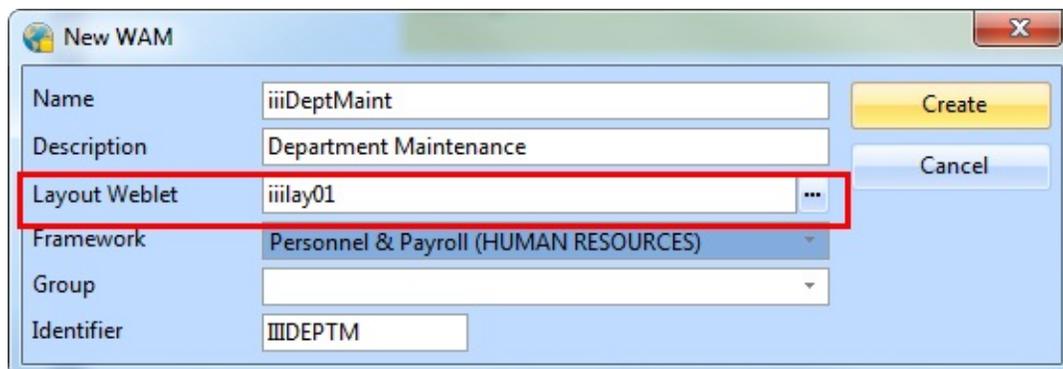
Contrary to what the name suggests, a WAM layout does not have to be made of visual elements - although it usually is.

The non-visual elements of a layout include references to XSL documents for:

- Standard variables
- Standard data types
- Style
- JavaScript
- Default hidden fields

## What is a Layout Weblet?

A layout weblet is simply a special kind of weblet which contains the XSL and XHTML which together with appropriate cascading style sheets, defines the content and appearance of a web page. Once you have created and tested your own layout weblet, you can use it as a common layout when creating each WAM which makes up your application.



Name	iiiDeptMaint	Create
Description	Department Maintenance	
Layout Weblet	iiiIay01	Cancel
Framework	Personnel & Payroll (HUMAN RESOURCES)	
Group		
Identifier	IIIIDPTM	

## What do Layouts Determine / Control?

The layout is a key element in the generated webroutine presentation. It ensures that a consistent interface is available across WebRoutines.

When you view your WebRoutine in the LANSAs Editor's Outline tab, the layout weblet is generally at the highest level in the outline tree. This indicates that all weblets below the layout in the tree can refer to the documents specified in the layout weblet.

Some of the things controlled by layouts include:

- The appearance of any menus
- Available menu options
- The appearance of a message box
- The Cascading Style Sheet to be applied

- Access to common JavaScript functions
- Definition of any global hidden fields
- Standard variable definitions which may be referenced in other weblets
- Whether a visual layout should be applied.

Of course if you define your own layout, you can decide what common elements need to be included in the interface.

## **How is a WAM Layout assigned to a WAM?**

A WAM-specific layout weblet is automatically generated for a WAM the first time it is built or compiled unless one already exists.

By default, when XSL is generated, the processor checks if a WAM-specific layout weblet already exists for the WAM. If a WAM layout does not exist, a new WAM layout weblet is generated and stored in the repository where the name is composed of the WAM Identifier followed by "\_layout". After it has been generated, your WAM-specific layout weblet is referenced by all the web routines in the associated WAM. Any changes to the WAM-specific layout weblet will be reflected in all of the WAM's web routines.

The *Generate XSL* options on a WAM compilation do not regenerate the WAM-specific layout. A WAM-specific layout is generated only once. Any subsequent modifications to the WAM-specific layout, or the assignment of a different layout, must be performed in the LANSAs Editor.

## **How Do I Create My Own Site Layout?**

As you have already seen in exercise WAM025, you can use the *Web Application Layout Manager Wizard* to create a common layout based on one of the three main designs supplied, and choose one of nine themes to control background and foreground colours and fonts used.

To create a layout weblet from scratch, select the Weblet option in the LANSAs Editor's New toolbar button dropdown list. In the dialog select the Layout Weblet option to create a Layout weblet.

Alternatively, you can copy one of the shipped layout weblets as a basis to creating your own.

In this exercise you will begin by creating your "company web page" layout outside of LANSAs, together with a style sheet to control its appearance. You will then build a layout template based on this company layout and test it by building a simple WAM which uses it.

To achieve these objectives, you will complete the following:

Step 1. Create a Simple Company Test Layout

Step 2. Create a Layout Template

Step 3. Refine Layout Weblet Definition

Step 4. Test the New Layout

Step 5. Review Structure of Layout XSL and HTML

Summary

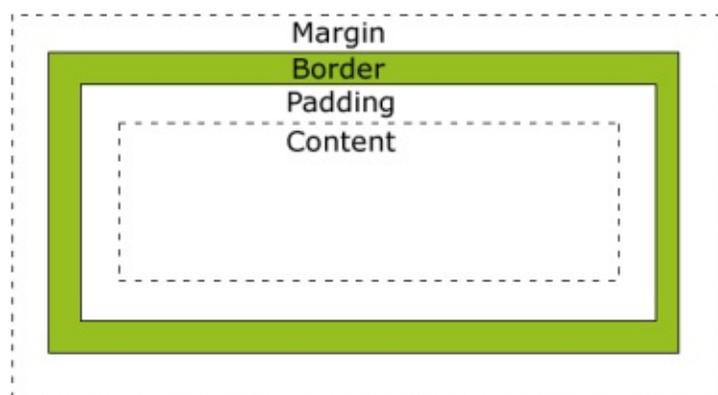
## Step 1. Create a Simple Company Test Layout

The layout you will define will be constructed using DIVs and CSS. You could also define a page layout using tables. If you search the Internet on this topic, you will find a number of forums where the pro's and con's are discussed at great length!

### The CSS Box Model

It's important to understand this concept. See [www.w3schools.com](http://www.w3schools.com) for more information.

The image following illustrates the box model:



The different parts are:

- **Margin** - Clears an area around the border. The margin does not have a background color, it is completely transparent
- **Border** - A border that goes around the padding and content. The border is affected by the background color of the box
- **Padding** - Clears an area around the content. The padding is affected by the background color of the box
- **Content** - The content of the box, where text and images appear

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

If the following is the CSS applied to an HTML element:

```
width:250px;  
padding:10px;  
border:5px solid gray;  
margin:10px;
```

The total width of the element is 300px, calculated as:

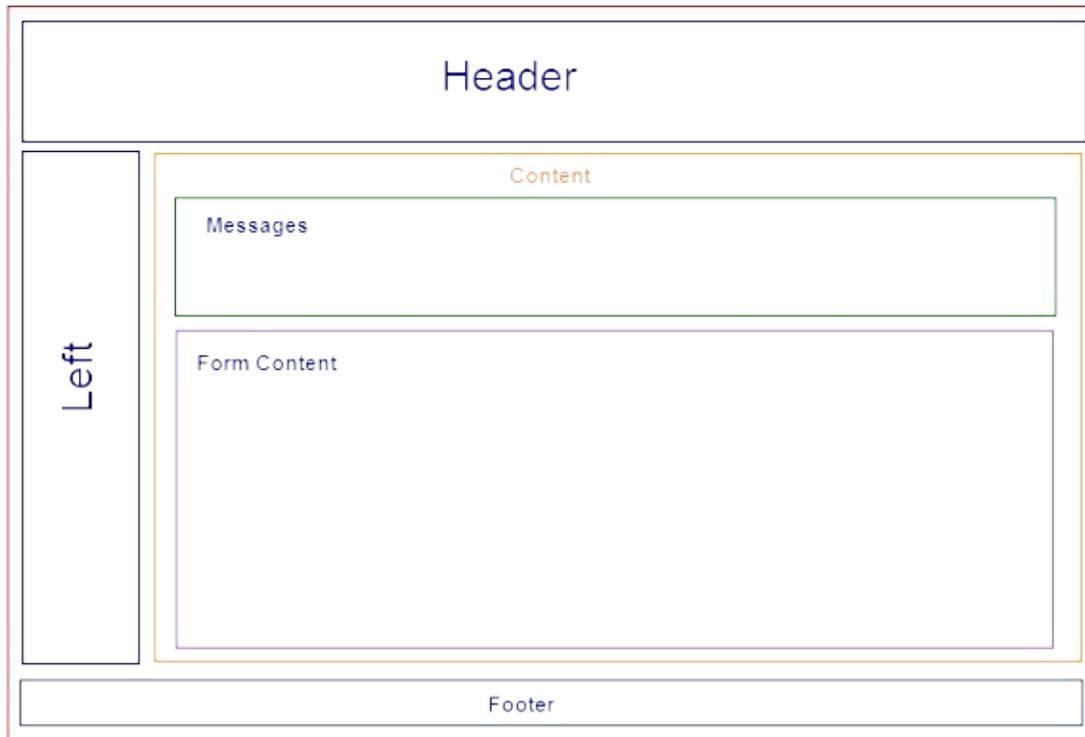
250px (width)  
+ 20px (left and right padding)  
+ 10px (left and right border)  
+ 20px (left and right margin)  
= 300px

## Acme Company Layout Design

The completed layout will look like the following:



This layout will be constructed as follows:



Each area is defined by a DIV with a unique id. A stylesheet (CSS) will define the position, size and appearance of each DIV. Other content is defined within the appropriate DIV. The content area DIVs are nested so that content, contains messages and form content.

1. Create a new folder in \My Documents called **iii\_layout**.
2. Copy the following code into Notepad and save it into \My Documents\iii\_layout as a file name **iii\_acme\_layout.html**.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title> LANSA | Training | cross browser fixed header/footer/left column
layout scrolling middle area | WAMs | </title>
<link REL="stylesheet" TYPE="text/css" href="iii_acme_style.css">
</head>

<body class="acme_layout">
<div id="acme_header">Heading</div>
```

```
<div id="acme_footer">footer</div>
```

```
<div id="acme_sidebar"> <div style="padding-top:400px">Left Panel</div>  
</div>
```

```
<div id="acme_content"> <h2>Content Area</h2>
```

```
<div id="acme_messagesContainer">
```

```
<h2>Messages</h2>
```

```
</div>
```

```
<h2>Form Content</h2>
```

```
</div>
```

```
</body>
```

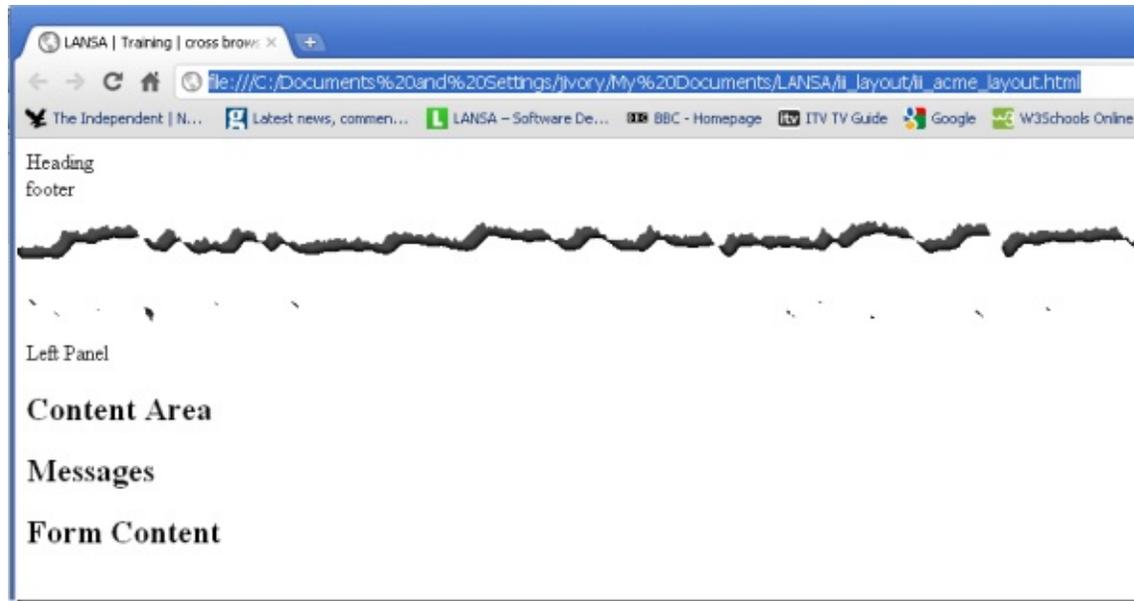
```
</html>
```

3. Change the style sheet named in this layout as **iii\_acme\_style.css** to use your initials.

Use the *Save as type* **All** to save the file with the correct extension.

4. Review the contents of this HTML file.
  - a. The `<!--DOCTYPE` puts IE into standard mode.
  - b. Inside the `<HTML` tag the page content is declared as XHTML and the primary language is English (EN).
  - c. The character set for the document is declared inside the `<meta` tag.
  - d. The `<link` tag declares a style sheet file (CSS) to be referenced by this HTML document. As this does not currently exist, your web page will have no formatting, except browser defaults.
  - e. The web page content is defined as DIVs within the `<body></body>` tags.
  - f. In each DIV, text wrapped by `<h2>` tags to identify each area.
  - g. Each DIV has an **id**, for example `<div id="acme_content_column"></div>`
  - i. The messages DIV is defined inside the content div.
5. Use Windows Explorer to navigate to `\My Documents\iii_layout` and double click on the file `iii_acme_layout.html` to open it in the default browser.

Your web page will currently look like the following:



Leave your web page open in the browser.

6. Copy the following code into Notepad and use the *Save as type: All* option to save it as **iii\_acme\_style.css**, to folder `\My Documents\iii_layout`.

```
#acme_content
{
overflow: auto;
position:absolute;
z-index:6;
top:160px;
bottom:50px;
left:200px;
right:0;
border: solid;
border-color: red;
padding: 10px;
}
html {
height:100%;
max-height:100%;
padding:0;
margin:0;
border:0;
font-family:georgia, palatino linotype, times new roman, serif;
overflow: hidden;
```

```
}
body
{
height:100%;
max-height:100%;
overflow:hidden;
padding:0;
margin:0;
border:0;
border: none;
}
#acme_header
{position:absolute;
margin:0;
top:0;
left:0;
display:block;
width:100%;
height:160px;
z-index:5;
overflow:hidden;
font-family:georgia, palatino linotype, times new roman, serif;
border : solid;
border-color: fuchsia;
}
#acme_footer
{
position:absolute;
margin:0;
bottom:0;
left:0;
display:block;
width:100%;
height:50px;
font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
z-index:5;
overflow:hidden;
border: solid;
border-color: aqua;
```

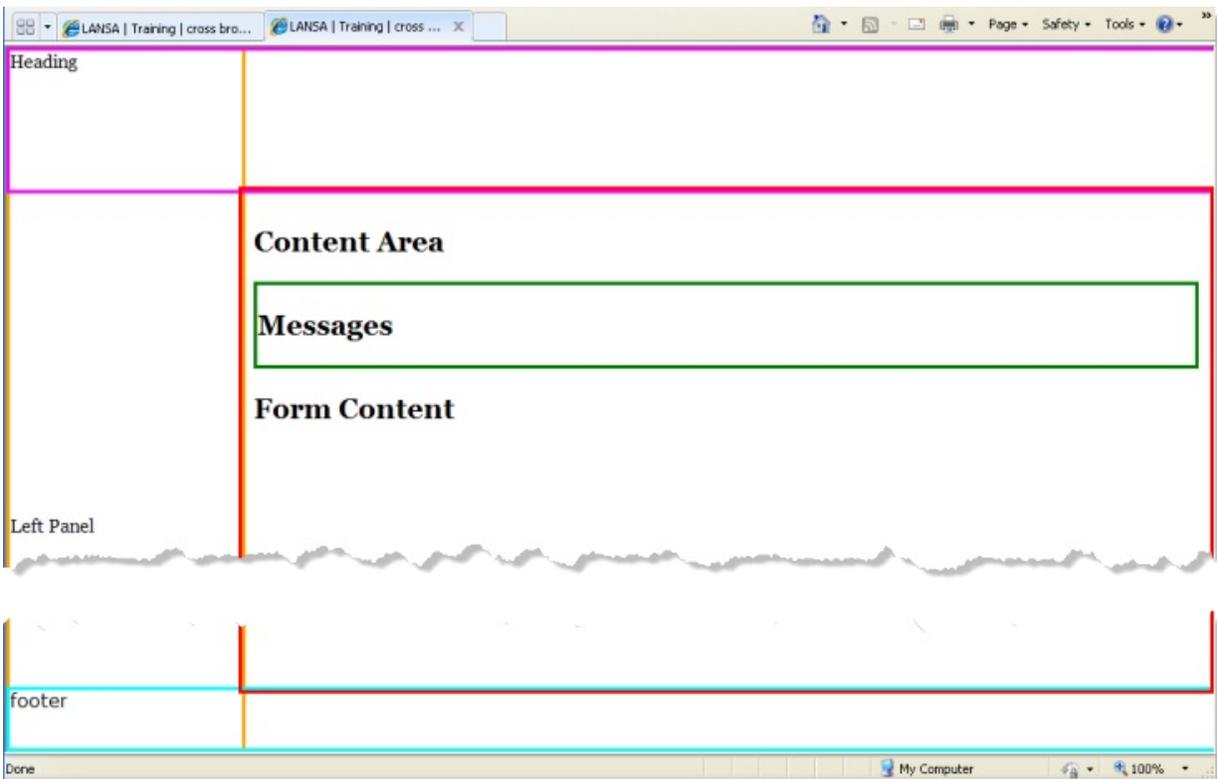
```
}
#acme_sidebar
{
position:absolute;
left:0;
top:0;
bottom: 0;
width:200px;
height : 100%;
z-index:4;
border: solid;
border-color: orange;
}
#acme_content p
{
padding:10px;
}
.bold {font-size:1.2em; font-weight:bold;}
```

```
dd {display:none;}
.p1
{
font-size: .4em;
color: white;
}
#acme_messagesContainer
{
top:0px;
min-height: 50px;
left: 0px;
overflow: auto;
z-index: 6;
border: solid;
border-color: green;
}
```

Notepad is not an ideal editor for CSS files. For your own work, we recommend you use a proper stylesheet editor such as *TopStyle*.

Leave the CSS file open in Notepad.

7. Review the styles sheet, which contains:
  - a. Styles for each id used in your web page, for example `#acme_content{overflow : auto; . . . . . }`
  - b. The head, foot, sidebar and content DIVs are all position absolutely.
  - c. **acme\_head** is positioned at the top of the page (`top : 0; left : 0;`)
  - d. **acme\_content** is positioned below **acme\_head** and to the right of **acme\_sidebar** (`top : 120px; left 200px;`)
  - e. The `acme_head`, `acme_foot`, `acme_sidebar`, `acme_content` and `acme_messagesContainer` DIVs each have a solid coloured border (just for the moment, to make their position visible).
  - f. `acme_content` has a setting of `overflow : auto`. This will provide a scrollbar if the content exceeds the space available.
8. Refresh your web page which should still be open in the browser. It should look like the following:



9. Make the following changes to your Stylesheet file (CSS)
  - a. Remove the border and border-color styles from `acme_head`, `acme_foot`, `acme_sidebar`, `acme_content` and `acme_messagesContainer`.

b. Make the following additions to the styles shown:

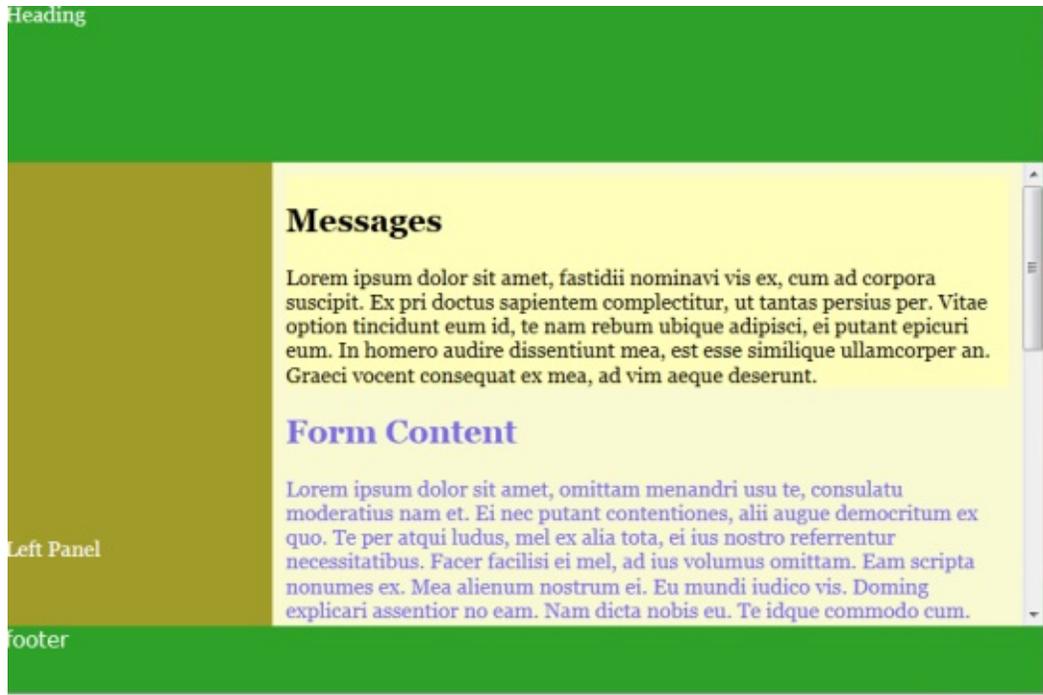
```
#acme_content
{
Background : #fafad2;
Color : #7b68ee;
}
#acme_header
{
Background : #2ea129;
Color : #fff;
}
#acme_footer
{
Background : #2ea129;
Color : #fff;
}
#acme_sidebar
{
Background : #a19c29;
Color : #fff;
}
#acme_messagesContainer
{
Background : #ffffbb;
Color : black;
}
```

c. Save your style sheet.

10. Refresh your web page in the browser. It should now look like the following:



11. Go to the following web site: <http://generator.lorem-ipsuam.info/>
  - a. Using Notepad to edit your web page HTML document, copy and paste text from the lorem-ipsuam web site into the **messages** DIV and the **content** DIV below the messages DIV and the *Form Content* text. Paste enough text to overflow the content area in the browser.
  - b. Remove the Content Area text (and H2 tags) from the top of the **content** DIV.
  - c. Refresh your web page in the browser. It should now look like the following:



Note that the content area has been given a vertical scroll bar.

## Step 2. Create a Layout Template

In this step you will create a new layout weblet. You will see that the "create new layout weblet" function provides a default layout. You will remove much of this content and replace it with code based on the test layout which you created in Step 1.

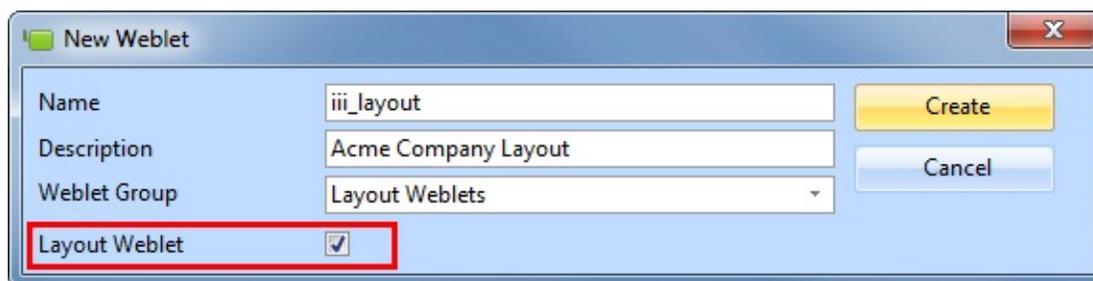
1. Use the *New / Weblet* dialog to create a layout weblet.

*Name:* **iii\_layout**

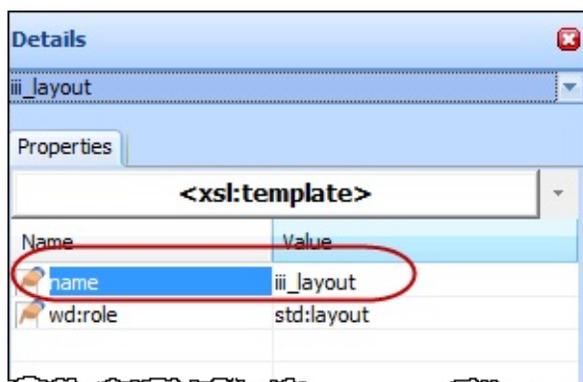
*Description:* **Acme Company Layout**

*Weblet Group:* **Layout Weblets**

Select the *Layout Weblet* checkbox.



2. The new layout weblet will open in the editor. In the *Design* view, click above the top of the layout and press F7. Select the *Details* tab to display the layout weblet properties.
  - a. Change the name to **iii\_layout**.



- b. Save the change.
3. Select the *XSL* tab and change the `<wd:template name` to **iii\_layout**.

```

<wd:template name="iii_layout">
  <wd:description icon="icons/std_layout.ico">
    <wd:name lang="ENG">New Layout</wd:name>
  </wd:description>
</wd:template>
<!-- Change your layout weblet name -->

<xsl:template name="iii_layout" wd:role="std:layout">
  <xsl:param name="window_title"
    select="$lweb_context/lxml:webapplication-title"

```

4. Change the `<wd:description. . . <wd:name` to **Acme Layout**.

```

<wd:template name="iii_layout">
  <wd:description icon="icons/std_layout.ico">
    <wd:name lang="ENG">Acme Layout</wd:name>
  </wd:description>
</wd:template>

```

5. As you continue to edit the XSL, regularly save your changes.

6. As you edit the XSL, the syntax will be checked, and any errors highlighted, for example:

```

<wd:definition>
  <wd:default-theme>
    <wd:style name="XWT01J" />
    <wd:style name="XWT01L" />
  </wd:default-theme>
  <wd:group name="Layout Weblets" />
</wd:definition>
<wd:template name="iii_layout">
  <wd:description icon="icons/std_layout.ico">
    <wd:name lang="ENG">iii Acme Layout</wd:name>
  </wd:description>
</wd:template>
<!-- Change your layout weblet name -->

```

Missing <

7. Delete the inline style declaration `<style></style>`. Delete the style tags and everything between them.

This block of code begins, and occupies 29 lines:

```

<style type="text/css">
#lpage_container {

```

8. Towards the end of the XSL there are 7 `<xsl:templates. . .>` blocks of code, which need to be deleted.

They begin **below** the main template `<xsl:template name="iii_layout:.. . .></xsl:template>` definition:

```

<xsl:template match="/lxml:data">
<xsl:template match="/lxml:data" mode="content.sidebar1">
<xsl:template match="/lxml:data" mode="content.sidebar2">
<xsl:template match="/lxml:data" mode="content.header">
<xsl:template match="/lxml:data" mode="content.navigation">
<xsl:template match="/lxml:data" mode="content.footer">
<xsl:template match="/lxml:data" mode="content.hidden">
  Delete from the lines listed, up to and including their end tag
</xsl:template>.

```

**Note:** Make sure you do not delete the `</xsl:transform>` at the end of the document.

9. Save your changes.
10. Find the code shown below and mark it for retention by adding the comment lines shown around it.

New code is highlighted in red, italic.

Hint: Search (Ctrl+F) for the first line shown using `lpage_content_column`. The other lines shown follow immediately after this.

```

<!-- KEEP -->
<div id="lpage_content_column">
<div id="lpage_content" class="lpage_content_area">
<xsl:if test="$show_title">
<h2 class="title">
<xsl:value-of select="$title_text" />
</h2>
</xsl:if>
<div wd:content="content">
<xsl:apply-templates select="*" />
</div>
</div>
</div>
<!-- END KEEP -->

```

11. **Above** the code shown in 10. locate the following line. Delete this line, and everything up to your first `<!--keep -->` comment line.

```

<div id="lpage_header" wd:content="content.header">

```

**Hint:** Move the cursor to the top of the XSL document and search (Ctrl+F) for `lpage_header`.

12. The last step will have highlighted as an error, the `</div>` tag below your `<!--end keep-->` comment line. Delete this `</div>` and everything before:

```
</xsl:element>
```

**Note:** Do not delete the line `</xsl:element>`.

13. Save your changes. You should have no errors at this point.

14. With your new web page layout (created in step 1) open in Notepad:

a. Remove the text which you added from the web site:

<http://generator.lorem-ipsuam.info/>.

b. Copy all the code inside the `<body></body>` tags (not including the `<body>` `</body>` tags) and paste in into your new WAM layout within the `<body>` `</body>` tags. These are **below** the `</xsl:if>` and before `<!--KEEP -->`. Your code should look like the following.

Inserted code is shown in red.

```
</xsl:if>
```

```
<div id="acme_header">Heading</div>
<div id="acme_footer">footer</div>
<div id="acme_sidebar">
<div style="padding-top:400px">Left Panel</div>
</div>
<div id="acme_content">
<div id="acme_messagesContainer">
<h2>Messages</h2>
</div>
<h2>Form Content</h2>
</div>
```

```
<!-- keep -->
```

15. Change the *class* of the `<body>` tag to **acme\_layout**. This line should now look like the following:

```
<body class="acme_layout">
```

16. Move the code (cut and paste) shown in red below, from within the block of xsl which you commented to keep, into the position shown, immediately before the `<h2>Form Content</h2>` line.

**Move only the code shown in red.**

```
div id="acme_header">Heading</div>
<div id="acme_footer">footer</div>
<div id="acme_sidebar">
<div style="padding-top:400px">Left Panel</div>
</div>
<div id="acme_content">
<div id="acme_messagesContainer">
<h2>Messages</h2>
</div>
```

```
<xsl:if test="$show_title">
<h2 class="title">
<xsl:value-of select="$title_text" />
</h2>
</xsl:if>
```

```
<h2>Form Content</h2>
</div>
```

17. Move the code shown in red below from the xsl you commented to keep, into the position shown, below the `<h2>Form Content</h2>` line. The moved code is shown in red.

```
<div id="acme_header">Heading</div>
<div id="acme_footer">footer</div>
<div id="acme_sidebar">
<div style="padding-top:400px">Left Panel</div>
</div>
<div id="acme_content">
<div id="acme_messagesContainer">
<h2>Messages</h2>
</div>
<xsl:if test="$show_title">
```

```
<h2 class="title">  
<xsl:value-of select="$title_text" />  
</h2>  
</xsl:if>  
<h2>Form Content</h2>
```

```
<xsl:apply-templates select="*" />  
</div>
```

18. Delete the remaining saved xsl. i.e everything which now remains within the comments `<!--keep -->` and `<!--end keep -->`.
19. Save your changes. At this point you have your basic layout defined. No stylesheet is associated with it except for the styles defined by the `standard_style.xsl` weblet.

### Step 3. Refine Layout Weblet Definition

In this step you will define a style *External Resource* for your new style sheet, and add the external resource to your layout.

1. You created a style sheet (iii\_acme\_style.css) for the new layout in *Step 1*.
  - a. Copy your iii\_acme\_style.css from \My Documents\iii\_layout to your Visual LANSA web server \images\style folder. The path name should be similar to:

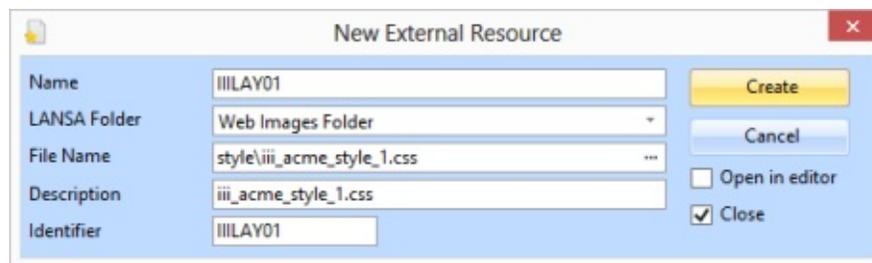
C:\Program Files (x86)\LANSA\WebServer\Images\style

On the IBM i, copy to the web server /images/style folder for the LANSA installation which you are using. The path name will be similar to:

LANSA\_<PGMLIB>/webserver/images/style

Where <PGMLIB> is your LANSA program library.

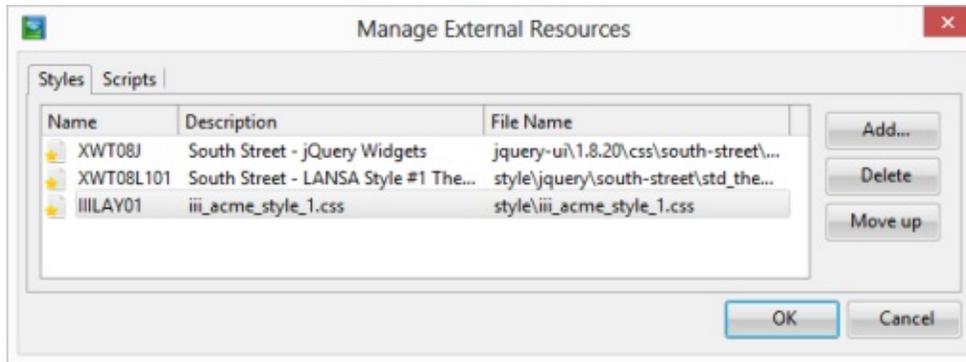
- b. Create an *External Resource* for this style sheet using the *File* menu / *New / External Resource / External Resource* option.



Enter the *External Resource Name*: **IIILAY01** (using your initials) and use the *Ellipsis* button for the *File Name* to select your style sheet file. The *LANSA Folder* and *Description* will be automatically filled.

- c. Click the *Create* button to save the new *External Resource* definition. You do not need to open it in the editor.
2. With your layout weblet open in the *Design view*, add the *External Resource* from the design ribbon *External Resource* option.

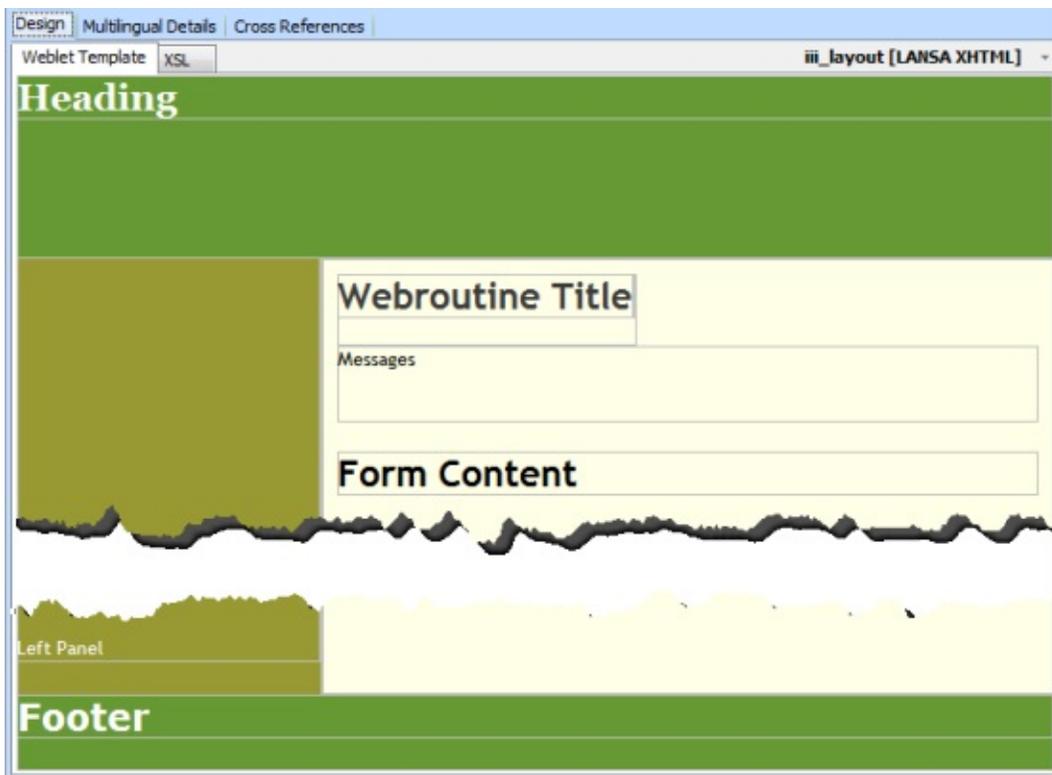
At the same time also add *External Resources* for styles XWT08J and XWT08L101. These will add style sheets to set fonts, colors and background colors for your page contents to blend with the green header and footer being used by the layout.



3. Select the XSL tab and delete the default theme external resources which were part of the default layout.

```
<wd:style name="XWT01J" />
<wd:style name="XWT01L" />
```

4. Save your changes. Your layout should now look like the following:



5. Remove the following text:

a. Delete this line, which was defined with a <div> including an inline style, in order to position the text where it would be shown and not hidden:

```
<div style="padding-top:400px">Left Panel</div>
```

b. Delete the following text, including any header tags associated with it (e.g. `<h2></h2>`)

Messages

Form Content

6. In the `acme_header` DIV replace the "Heading" text with the following. This may be copied from *WAM Tutorials* in the online guide.

```
<span style="padding-left: 10px">  
<h1>Acme IT Services</h1>  
</span>  
<br />  
<span style="padding-left: 10px">  
<h2>Software for Business</h2>  
</span>
```

7. In the `acme_footer` DIV replace the "footer" text with:

```
<span style="padding-left: 10px; padding-top: 5px">© Acme Ltd 2011</span>
```

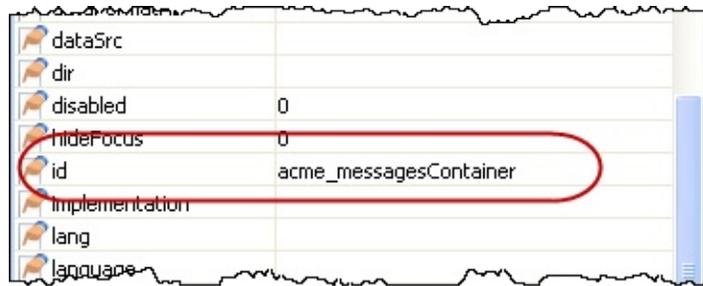
These changes use the `<span>` tag to position the text using inline styles. A better solution would be to give these span tags an id and control the appearance via the external stylesheet.

8. Save your changes and review your layout in the *Design View*.

9. The DIV `acme_messagesContainer` is shown in the *Design* view as an outline box, above the webroutine title text. Select this area:



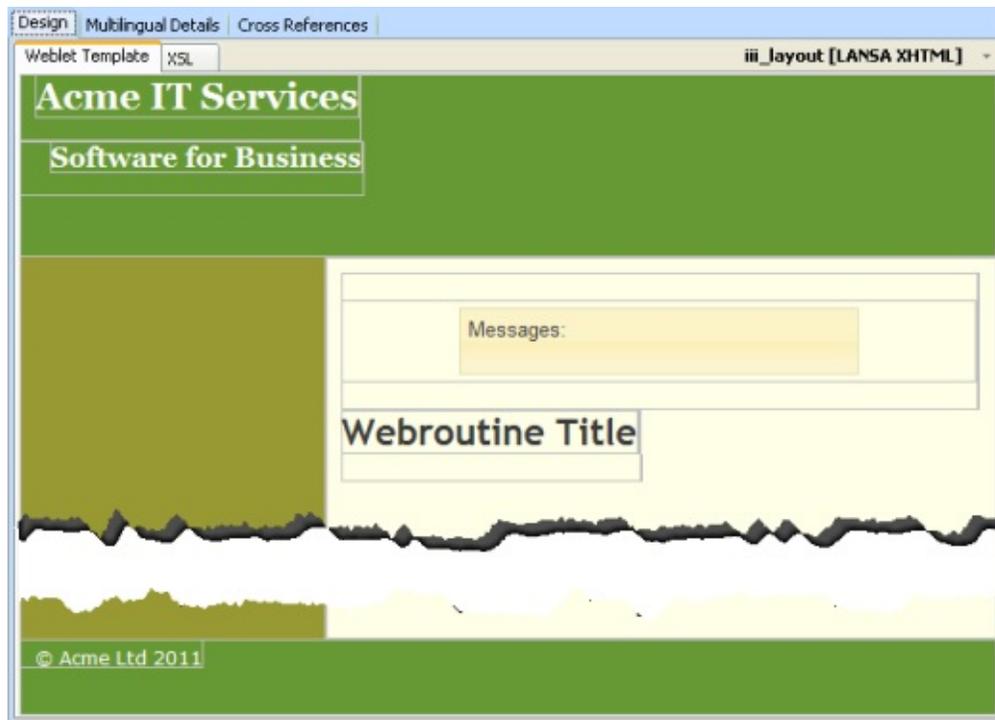
Confirm you have the correct element selected by examining the *Details* tab, and checking the *id*.



From the *Standard Weblets* group, drop a messages weblet into the acme\_messagesContainer DIV.

10. Save your changes

Your design should look like the following:



11. On the *Favorites/Weblet Templates* tab, select jQuery UI group from the top combo box.

The acme\_header DIV is the top green box containing the text "Acme IT Services"

- a. Select the acme\_header DIV
- b. Drop a *jQuery UI Menubar* into the **acme\_header** DIV.
- c. Select the menu bar, and use the right mouse on one of the "handles" to *Insert HTML / Div*. This will insert a DIV around the menu bar, which can

then be given an *id*. An addition to the style sheet for this id, will then position the menu bar.

- d. With the new DIV selected, select the *Details* tab and change the *id* to **lpage\_navBar**.
- e. Add the following style to your style sheet *iii\_acme\_style.css* and save the style file. This will override the LANSA styles for the *id* **lpage\_navBar**.

```
#lpage_navBar
{
min-width: 396px;
min-height: 44px;
position: absolute;
top: 10px;
left: 550px;
}
```

- f. Save the changes to your layout *iii\_layout*.
  - g. From the *Design* ribbon use the *Refresh / Web Browser* option to refresh the image in the *Design* view. The menu bar position in the header area should now reflect the CSS changes you added for *lpage\_navBar*.
12. With the menubar selected, set up the *menuitems* property on the *Details* sheet, using the *Ellipsis* button button to open the *Design of...* dialog. Define a top level menu only as:

**Menu Item    Action URL**

Home	<a href="http://www.lansa.com">http://www.lansa.com</a>
Support	<a href="http://www.lansa.com/support/index.htm">http://www.lansa.com/support/index.htm</a>
Contact Us	<a href="http://www.lansa.com/about/contactus.htm">http://www.lansa.com/about/contactus.htm</a>
About	<a href="http://www.lansa.com/about/index.htm">http://www.lansa.com/about/index.htm</a>

13. Save your changes. Your design should look like the following:



14. Your layout `xsl:template` currently contains some parameters which will not be used. Delete the following code:

```
<xsl:param name="width_type" select=""fluid""  
wd:type="std:layout_width_type" />  
<xsl:param name="width" select=""1000px"" wd:type="std:css_length_unit" />  
<xsl:param name="sidebar1_width" select=""20%"" wd:tip_id=""  
wd:type="std:css_length_unit" />  
<xsl:param name="content_width" select=""50%"" wd:tip_id=""  
wd:type="std:css_length_unit" />  
<xsl:param name="sidebar2_width" select=""30%"" wd:tip_id=""  
wd:type="std:css_length_unit" />
```

## Step 4. Test the New Layout

1. Create a new WAM:

Name: **iiiTestLayout**

Description: **Test layout iii\_layout**

Layout weblet: **iii\_layout**

2. Add the following RDMLX code after the Begin\_com and compile the WAM.

```
Define Field(#empnow) Reffld(#empno)
Def_List Name(#empskills) Fields(#SKILCODE #GRADE #COMMENT #D/
Web_Map For(*both) Fields((#stdrentry *hidden))
Webroutine Name(begin) Desc('Employee Details and Skills')
Web_Map For(*both) Fields((#EMPNO *out) #SURNAME #GIVENAME #A
Case (#stdrentry)
When (= U)
#empno := #empnow
Update Fields(*all) In_File(pslmst) With_Key(#empno)
Otherwise
Select Fields(*all) From_File(pslmst)
Leave
Endselect
#empnow := #empno
Clr_List Named(#empskills)
Select Fields(#empskills) From_File(pslskl) With_Key(#empno)
Add_Entry To_List(#empskills)
Endselect
Endcase
Endroutine
```

3. Open the **begin** WebRoutine in the *Design* view.
  - a. Add a column to the employee fields table.
  - b. Drop a push button into the top of the new column
  - c. Remove the place holder characters.
  - d. Select the push button and set up its properties as:

**Property**

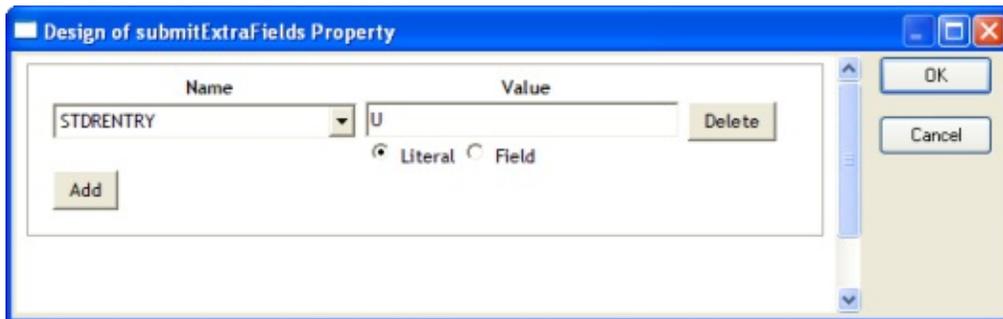
**Value**

Caption **Save**

On\_click\_wname **Begin**

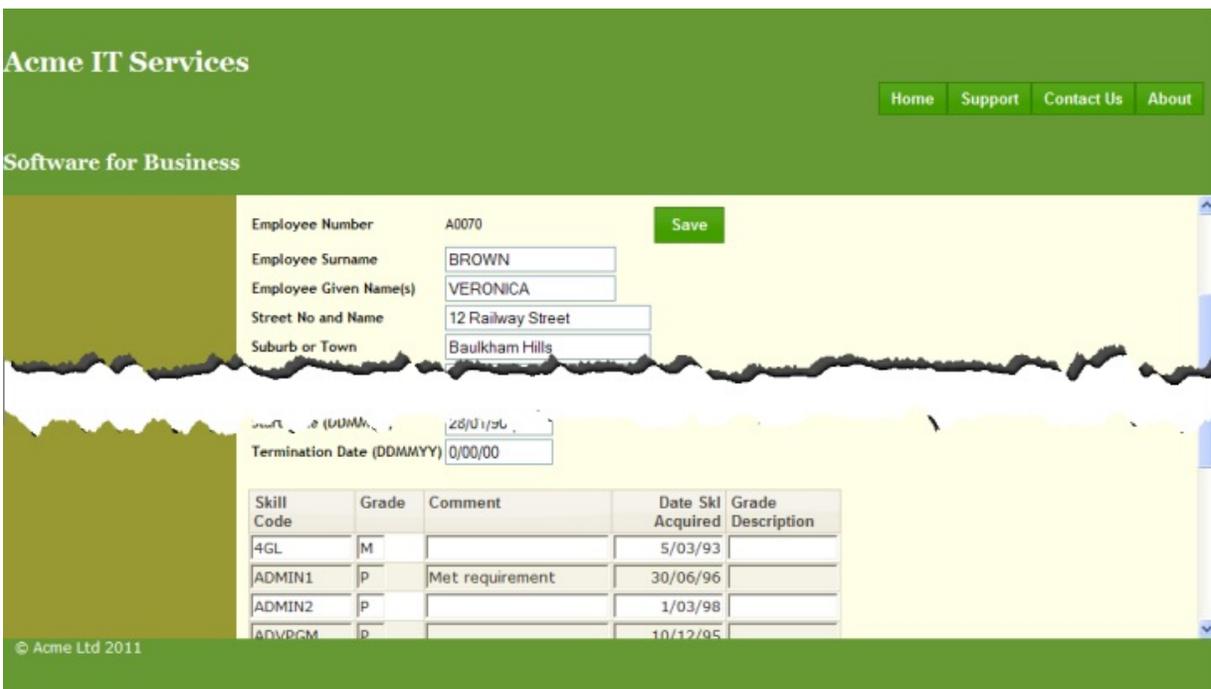
submitExtraFields **Field Name: STDREENTRY**  
**Literal Value: U**

e. Complete the submitExtraFields property using the Ellipsis button:



f. Save your changes.

g. Execute the WAM in the browser. Your web page should look like the following:



4. Clear the surname and given name fields and click the Save button.

Validation errors should be displayed in the messages weblet at the top of the content area, as shown:



The screenshot shows a web form titled "Employee Details and Skills" with a yellow background. At the top, a "Messages:" box contains two error messages: "A surname is required" and "Given name(s) must be specified". Below the messages, the form fields are: "Employee Number" with the value "A0070", "Employee Surname" with an empty text box, and "Employee Given Name(s)" with an empty text box. A green "Save" button is located to the right of the "Employee Number" field. The form is framed by a decorative, torn-edge border.

5. Try out the menu bar. Notice that the menu items appearance changes for hover and selection. This is controlled by the theme style sheet defined by the external resource XWT08L101.

## Step 5. Review Structure of Layout XSL and HTML

This step analyses the structure of the layout XSL, to explain what the main parts of the program are doing.

1. WAMs use the XSL Transformation language (XSLT) to process the XML output by a WebRoutine to produce a web page – an XHTML document. You can learn more about XSL at the [www.w3schools.com](http://www.w3schools.com) web site.

The first few lines of code, define the XSL namespace standards used in the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) 2003, 2011 LANSA          -->
<!-- XHTML WAM Layout           -->
<!-- $Revision:: 12              $ -->
<xsl:transform version="1.0" exclude-result-prefixes="lxml wd tsml"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data"
  xmlns:wd="http://www.lansa.com/2002/XSL/Weblet-Design"
  xmlns:tsml="http://www.lansa.com/2002/XML/Generation-
Metadata"
  xmlns="http://www.w3.org/1999/xhtml">
```

XSL code such as `<xsl:if . . .>` has a namespace of **xsl** and is a standard defined by the World Wide Web consortium.

XSL code such as `<wd:external-resources. . .>` has a namespace of **wd** and is LANSA defined standard.

2. The `<xsl:import . ./>` statements, import other weblets into this layout:

```
<!-- Standard imports          -->
<!-- Import the weblet XSL files used by your layout here -->
<!-- e.g.                      -->
<xsl:import href="std_variables.xsl" />
<xsl:import href="std_types.xsl" />
<xsl:import href="std_hidden.xsl" />
<xsl:import href="std_style_v2.xsl" />
<xsl:import href="std_script.xsl" />
<xsl:import href="std_messages.xsl" />
<xsl:import href="std_menubar.xsl" />
<xsl:output method="xml" omit-xml-declaration="yes" encoding="UTF-8"
```

```
indent="no" doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd" />
```

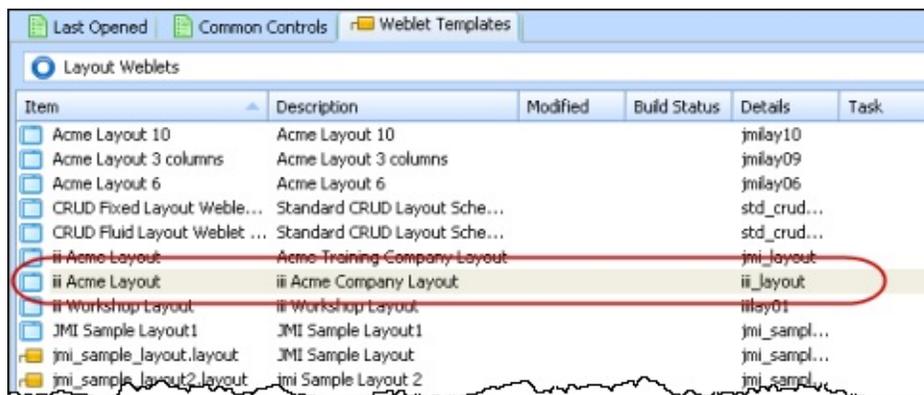
When the messages and menu bar weblets were added to this layout in the graphical editor, import statements for `std_messages.xsl` and `std_menubar.xls` were added.

3. The XSL statements which begin **wd:** are LANSa weblet design statements which are used within LANSa's graphical web page editor (actually an XSL editor). **wd:** is a LANSa namespace.

```
<wd:external-resources>
  <wd:style name="XWT08J" />
  <wd:style name="XWT08L101" />
  <wd:style name="IIILAY01" />
</wd:external-resources>
<wd:definition>
  <wd:default-theme />
  <wd:group name="Layout Weblets" />
</wd:definition>
<wd:template name="iii_layout">
  <wd:description icon="icons/std_layout.ico">
    <wd:name lang="ENG">iii Acme Layout</wd:name>
  </wd:description>
</wd:template>
```

The external resources will define links to external style sheets to be used by web pages based on this layout.

These tags also provide a name, description, group name and icon to be used for this weblet in the Repository.

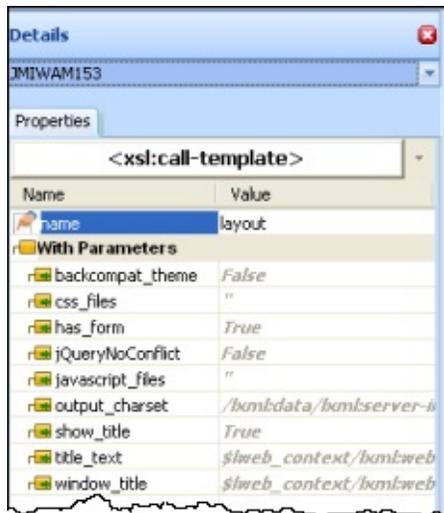


Item	Description	Modified	Build Status	Details	Task
Acme Layout 10	Acme Layout 10			jmlay10	
Acme Layout 3 columns	Acme Layout 3 columns			jmlay09	
Acme Layout 6	Acme Layout 6			jmlay06	
CRUD Fixed Layout Weblet...	Standard CRUD Layout Sche...			std_crud...	
CRUD Fluid Layout Weblet ...	Standard CRUD Layout Sche...			std_crud...	
ii Acme Layout	Acme Training Company Layout			jmi_layout	
iii Acme Layout	iii Acme Company Layout			ii_layout	
ii Workshop Layout	ii Workshop Layout			iiay01	
JMI Sample Layout1	JMI Sample Layout1			jmi_sampl...	
jmi_sample_layout.layout	JMI Sample Layout			jmi_sampl...	
jmi_sample_layout2.layout	jmi Sample Layout 2			jmi_sampl...	

4. xsl:templates are like subroutines. They can be called, passing parameters into the template.

```
<xsl:template name="iii_layout" wd:role="std:layout">
  <xsl:param name="window_title"
    select="$lweb_context/lxml:webapplication-title"
    wd:type="std:mtxt_variable" wd:tip_id="" />
  <xsl:param name="has_form" select="true()" wd:type="std:boolean"
    wd:tip_id="" />
  <xsl:param name="show_title" select="true()" wd:type="std:boolean"
    wd:tip_id="" />
  <xsl:param name="title_text" select="$lweb_context/lxml:webroutine-
title"
    wd:type="std:mtxt_variable" wd:tip_id="" />
  <xsl:param name="javascript_files" select="" wd:tip_id="" />
  <xsl:param name="jQueryNoConflict" select="false()" wd:type="std:boole
    wd:tip_id="" />
  <xsl:param name="css_files" select="" wd:tip_id="" />
  <xsl:param name="output_charset"
    select="/lxml:data/lxml:server-instructions/lxml:client-charset"
    wd:tip_id="" />
  <xsl:param name="backcompat_theme" select="false()" wd:type="std:boo
    wd:tip_id="" />
```

When you compile a WAM using this Layout Weblet, the generated WAM layout uses the layout weblet specified. When you then design a WebRoutines's web page, these parameters will be shown on the *Details* tab and could be set at design time and or at runtime.



5. The XHTML definition begins here. Note that an **xsl:if** is testing the value of an environment language variable (\$lweb\_ISO\_language). Further down another **xsl:if** refers to **\$window\_title** which is the value of a boolean input parameter.

```

<html>
  <xsl:if test="$lweb_ISO_language != "">
    <xsl:attribute name="xml:lang">
      <xsl:value-of select="$lweb_ISO_language" />
    </xsl:attribute>
    <xsl:attribute name="lang">
      <xsl:value-of select="$lweb_ISO_language" />
    </xsl:attribute>
  </xsl:if>
  <head>
    <title>
      <xsl:value-of select="$window_title" />
    </title>
    <xsl:choose>
      <xsl:when test="$backcompat_theme">
        <xsl:call-template name="style">
          <xsl:with-param name="theme_css_filename" select="" />
          <xsl:with-param name="css_files" select="$css_files" />
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="style">

```

```

        <xsl:with-param name="theme_css_filename" select=""none"" />
        <xsl:with-param name="css_files" select="$css_files" />
    </xsl:call-template>
</xsl:otherwise>
</xsl:choose>
<xsl:call-template name="script">
    <xsl:with-param name="javascript_files"
        select="$javascript_files" />
    <xsl:with-param name="trap_script_errors" select="false()" />
    <xsl:with-param name="jQueryNoConflict"
        select="$jQueryNoConflict" />
</xsl:call-template>
</head>

```

This code defines the contents of the head area of the web page. The XSL logic is generating XHTML based on the parameters provided at runtime.

6. Your page content is output within the HTML <body></body> tags.

```

<body class="acme_layout">
    <xsl:variable name="containerName">
        <xsl:choose>
            <xsl:when test="$has_form">
                <xsl:text>form</xsl:text>
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>div</xsl:text>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>
    <xsl:element name="{ $containerName }">
        <xsl:attribute name="id">lpage_container</xsl:attribute>
        <xsl:if test="$has_form">
            <xsl:attribute name="onsubmit">return _HandleDefaultSubmit(this
</xsl:attribute>
            <xsl:attribute name="method">post</xsl:attribute>
            <xsl:attribute name="name">LANSA</xsl:attribute>
            <xsl:attribute name="action">
                <xsl:value-of select="$lweb_context/lxml:action-request" />?
            </xsl:attribute>
            <xsl:call-template name="hidden_fields" />

```

</xsl:if>

The `xsl:choose` logic (like an RDML CASE loop) outputs a `<form>` tag or a `<div>` depending on the variable `$has_form`. **Has\_form** is a property of the layout, and is shown on the *Properties* sheet on the *Design* tab.

7. This section is largely the code which you provided and represents the design of your web page, when combined with the necessary stylesheet.

```
<div id="acme_header">
  <span style="margin-left:10px; padding-top:0px">
    <h1>Acme IT Services</h1>
  </span>
  <br />
  <span style="margin-left: 20px">
    <h2>Software for Business</h2>
  </span>
  <div id="lpage_navBar">
    <xsl:call-template name="std_menubar">
      <xsl:with-param name="menu_items"
        select="document('')/*/lxml:data/lxml:menu[@id='E4
  <xsl:with-param name="name"
    select="concat('o', position(), '_LANSA_17914')" />
    <!-- <xsl:with-param name="listname" select="?" /> -->
    <!-- <xsl:with-param name="orientation" select="?" /> -->
    <!-- <xsl:with-param name="show_arrows" select="?" /> -->
    <!-- <xsl:with-
param name="submit_selected_to" select="?" /> -->
    </xsl:call-template>
  </div>
</div>
<div id="acme_footer">
  <span style="margin-left: 10px; padding-
top:5px">© Acme Ltd 2011</span>
</div>
<div id="acme_sidebar"></div>
<div id="acme_content">
  <div id="acme_messagesContainer">
    <xsl:call-template name="messages">
      <!-- <xsl:with-
```

```
param name="target_window_name" select="?" /> -->
  </xsl:call-template>
</div>
<xsl:if test="$show_title">
  <h2 class="title">
    <xsl:value-of select="$title_text" />
  </h2>
</xsl:if>
<xsl:apply-templates select="*" />
</div>
</xsl:element>
</body>
```

This code also includes the menu bar and messages weblets which you added using the graphical editor.

The `<xsl:apply-templates select="*" />` outputs your WebRoutine fields and lists, together with output from any weblets which were added to the page, such as *Push Buttons*.

## Summary

### Important Observations

- You have created your own company layout which can be used as the Layout Weblet when creating your application WAMs
- Any changes you make to this common layout, will affect all WAM layouts which are based on this common layout.
- The shipped layout designs demonstrate how the same design can be significantly altered simply by changing the style sheets associated with the design. Consider this option, before embarking on development of a completely new layout.

### Tips & Techniques

- With the design tools now available, such as the *Microsoft IE Developer Toolbar* and a professional style sheet editor you can rapidly understand the detailed design of any web page you may want to adopt.

### What You Should Know

- How to create your own layout weblet.
- How to use CSS to control page layout which is based on DIVs.

## **WAM115 - Check in WAMs to IBM i**

### **Objectives**

- To check in and compile some of your exercise WAMs and their layouts to the IBM i
- To run the WAMs on the IBM i

This exercise is optional and will depend on the training facilities being used. Visual LANSA must be installed as a Slave Workstation to the IBM i LANSA Master.

All the files and most of the fields used in this workshop will already exist on the IBM i, in the partition being used for training.

In order to complete this exercise you will perform the following:

[Step 1. Check in a WAM and its Layout](#)

[Step 2. Run a WAM on the IBM i](#)

[Step 3. Run a WAM in debug on the IBM i](#)

[Summary](#)

### **Before You Begin**

This exercise assumes that you have completed all preceding exercises.

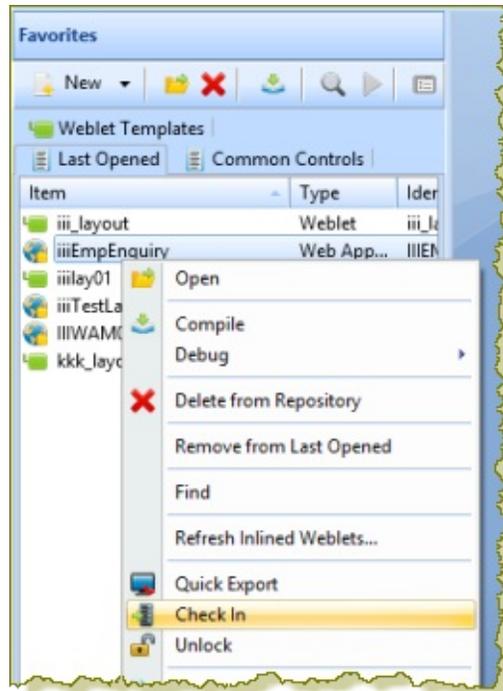
## Step 1. Check in a WAM and its Layout

To compile and run your WAM on the IBM i, you will need to ensure that you have also checked in any dependent objects.

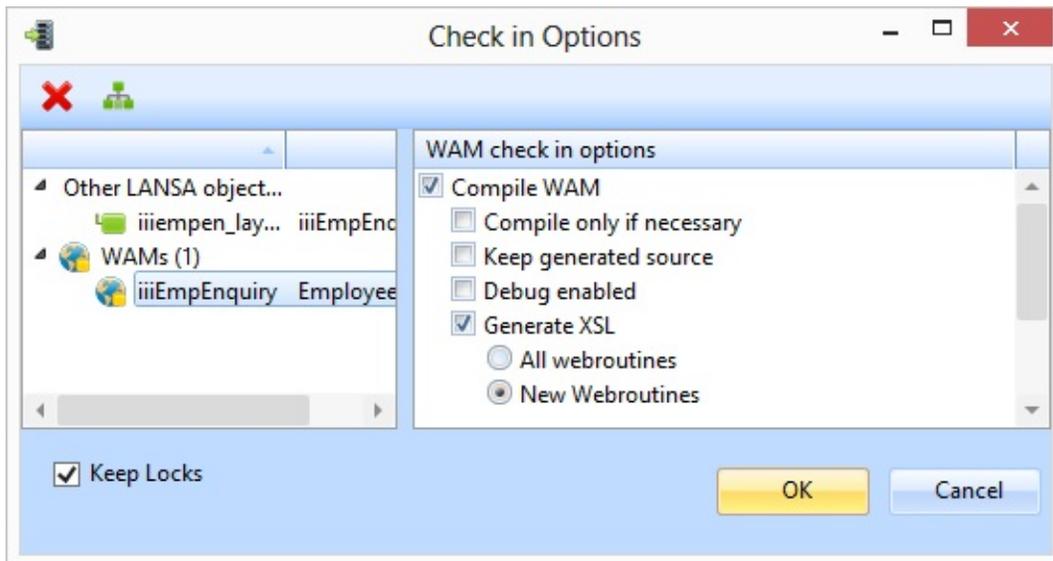
- Checking in a WAM also checks in the web page definition for each WebRoutine. The WAM layout will also be automatically checked in.
- You only need to check in a WAM layout once, or when it has been changed. You will find that LANS A V13 always checks in the WAM layout when you check in the WAM.
- When checking in a WAM, you should use the Cross References dialog to select and check in any dependent objects.
- If your WAM layout is based on a common company layout, you will check this layout in the first time you check in a WAM which uses it and whenever this common layout has been changed.
- Other definitions which you may need to consider are any custom weblets or *External Resources* which your WAM is using.
- You can check in using the *context menu* from the *Repository* tab, or from the *Last Opened* tab.
- With the WAM open in the editor, you can also check in from the *Home ribbon / Remote Systems/ Check in* button:



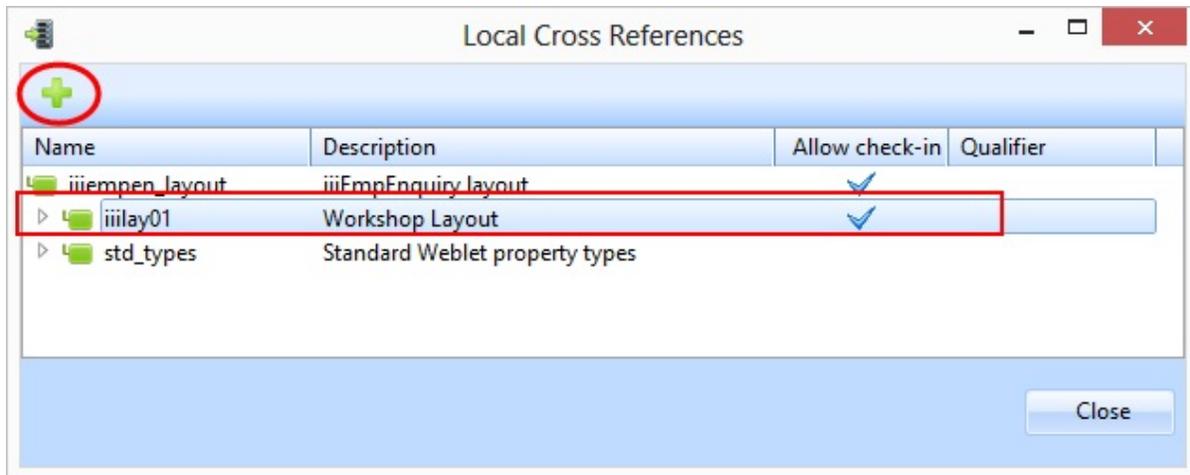
1. On the *Repository* tab find WAM iiiEmpEnquiry under *Web / Web Application Modules*. Use the context menu to *Check In*:



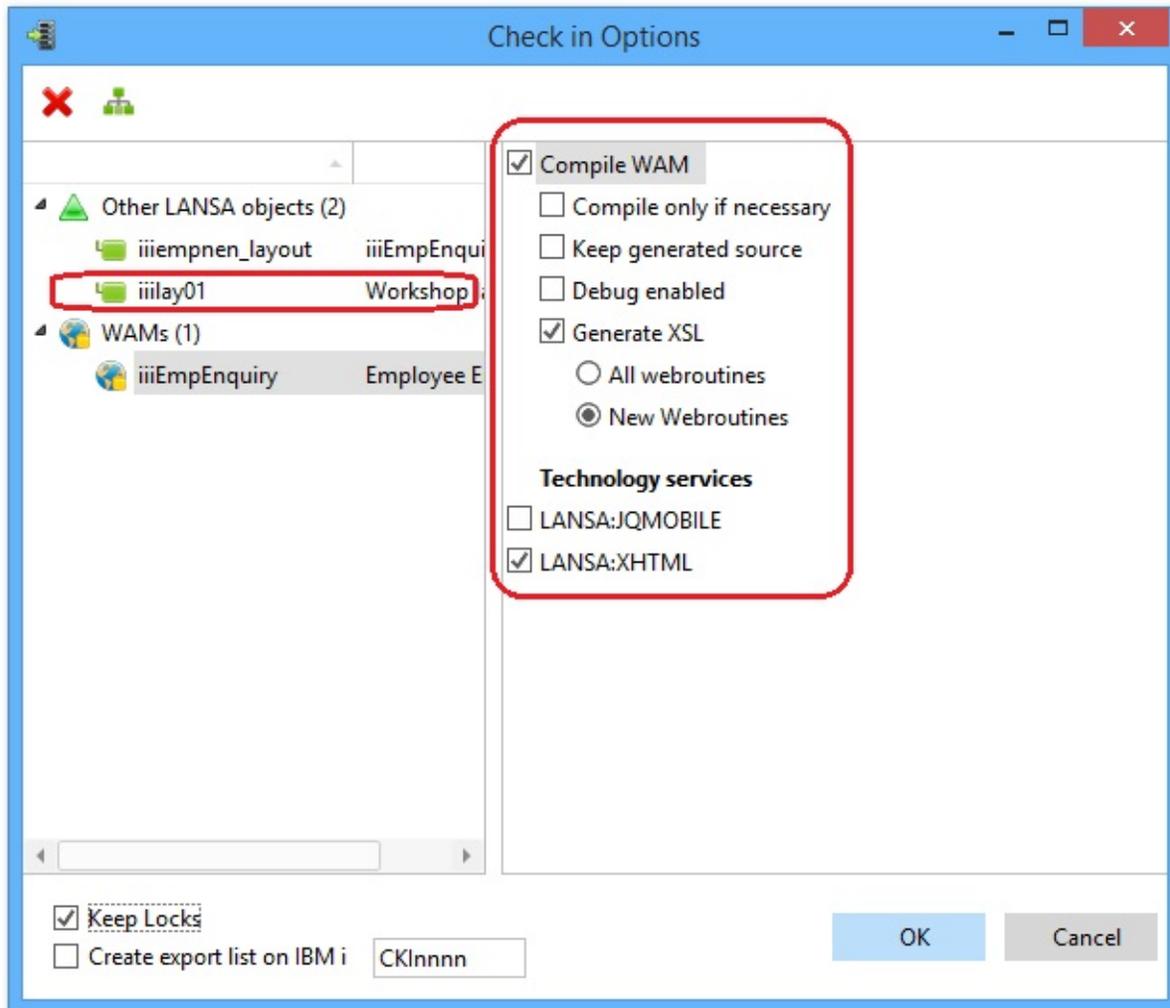
The *Check In Options* dialog opens.



2. Select the WAM layout to enable the  *Cross References* toolbar button. Click the toolbar button to open the *Local Cross References* dialog:



3. Select the common layout **iiilay01** and click the  *Add for check in* button, to add it to the check in list of objects.
4. Click the *Close* button to close this dialog. Your check in dialog should look like the following:



**Note:** The common layout **iiilay01** has been added to the objects to be check in.

Ensure that the *Check In* options are as shown.

When you check in a WAM for compile the following steps occur:

A C program for the WAM RDMLX code is generated and checked in.

A compile job is submitted on the IBM i.

The XSL Transformations for all WebRoutines are checked in for the chnology Services which are selected in the check in options.

The WAM layout is checked in.

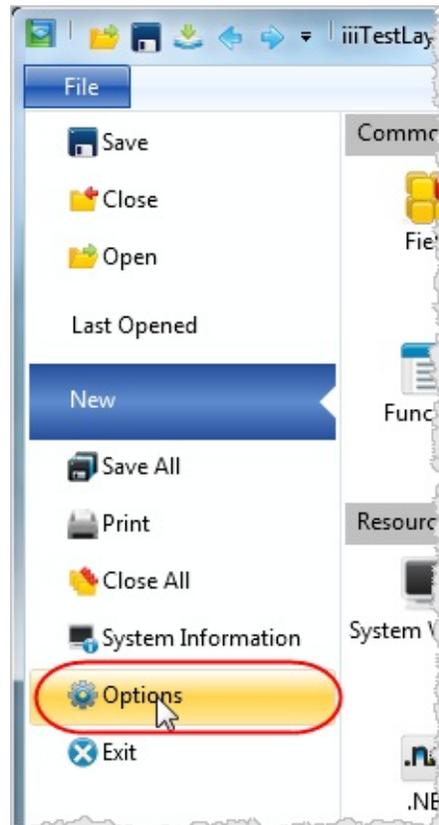
As noted, you need to also check in any other dependent objects as appropriate.

Your new WAM is locked to a Task Id. Usually this will be the Task Id you ected when you logged in the Visual LANSa. The lock is applied using your PC

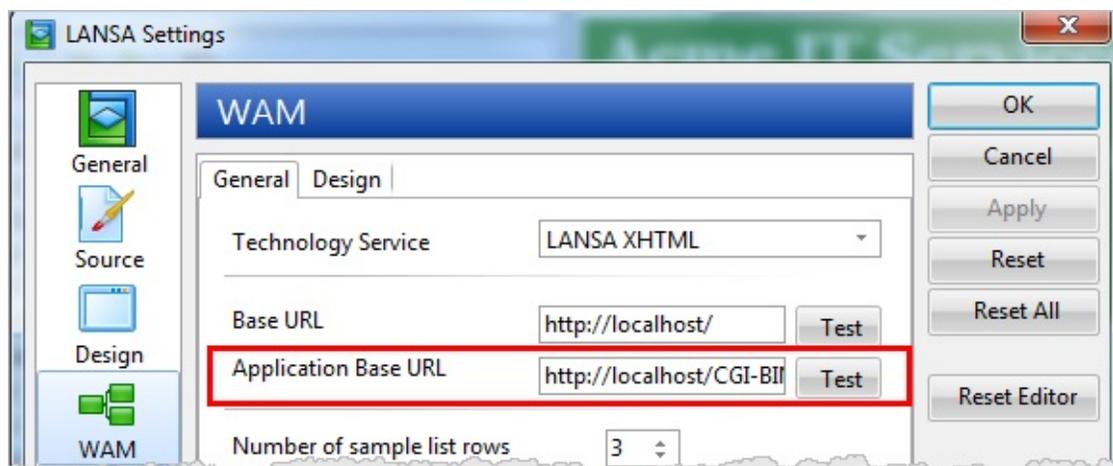


## Step 2. Run a WAM on the IBM ii

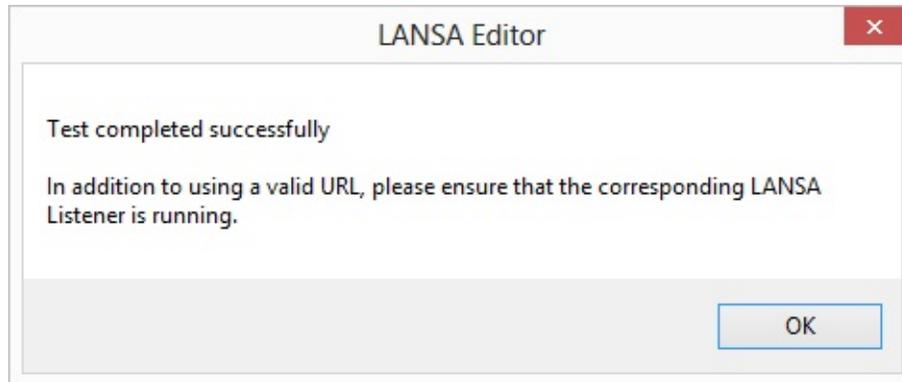
1. Open WAM iiiEmpEnquiry in the editor. Open the **begin** WebRoutine in the *Design* view.
2. Use the *Editor Options* from the *File* menu.



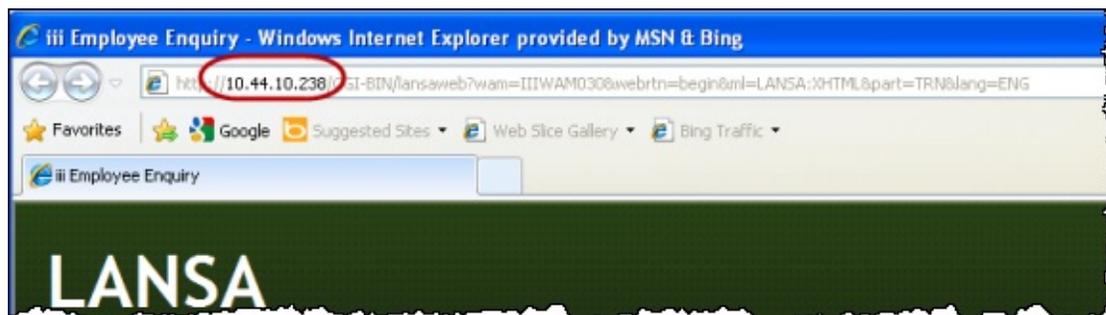
Select the WAM icon:



3. Change the *Application Base URL* setting to point to the IBM i server. This could be specified as a domain name or an IP Address.
4. Test the definition by clicking the *Test* button:



5. Close the settings dialog by clicking the *OK* button.
6. With the WAM *iiiEmpEnquiry* open in the editor, open the **begin** WebRoutine in the *Design* view and *Run* in the browser. Your WAM will be run on the IBM i.

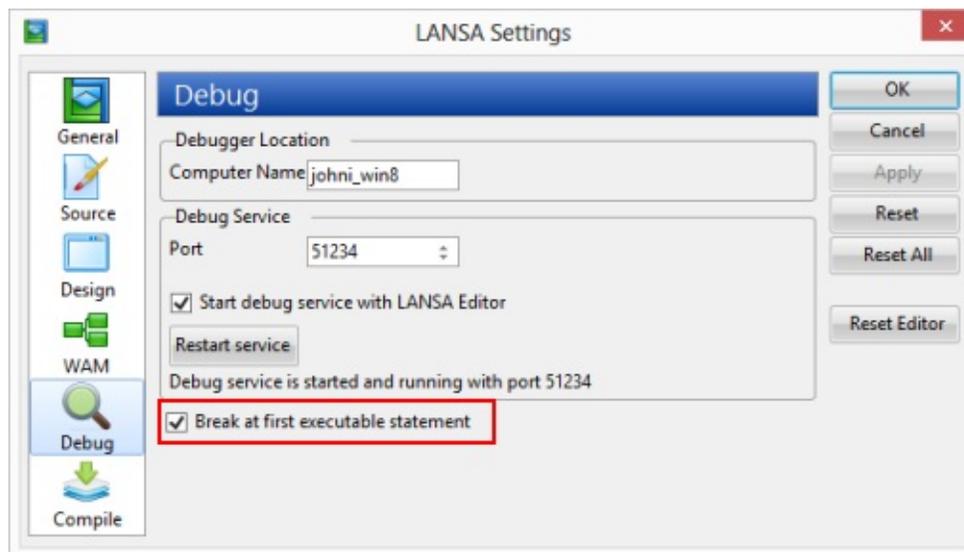


### Step 3. Run a WAM in debug on the IBM i

With your WAM checked in and compiled on the IBM i server, you can now run it in debug mode from Visual LANSA, while it is running on the IBM i server.

The *Base Application URL* settings must point to your IBM i server.

1. You should have WAM iiiEmpEnquiry open in the editor, with the **begin** WebRoutine open in the *Design* view. Check the debug settings from the *File / Editor Options*, select the *Debug* icon.

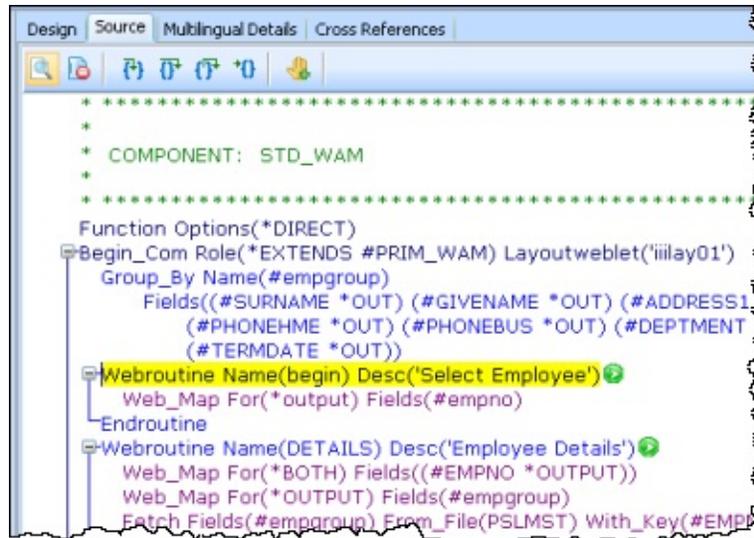


Ensure the *Break at first executable statement* option is selected. This means that initially you do not need to set any breakpoints in the program.



2. Run the begin WebRoutine using the  toolbar button.

The WebRoutine will execute in debug mode with the WAM executing on the IBM i server.



You can now continue to set breakpoints and use other debug features.

For further details, refer to the exercise [WAM055 - Using LANSa Debug](#) if required.

## Summary

### Important Observations

- In this exercise you selected a group of objects for check in by name. You can also select objects to check in:
  - By Task
  - By building a static Editor List or maintaining a dynamic editor list for your changes
  - By sorting the Repository tab by Date Modified and selecting changed objects
  - By reference to your Last Opened tab
- If you have changed design of a WebRoutine's web page, simply check in the WAM again. No recompile will be necessary
- If you have changed a common layout weblet, check this in and the WAM's which use it will include it at run time.
- If you have changed cascading styles, check in the *External Resource* which defines this style file.

### Tips & Techniques

- Remember that the details of a Field Visualization weblet are locked into the XSL when the WebRoutine web page is created and saved. If a field visualization has been changed, for example, from radio button group to dropdown, checking in the WAM will make your application behave correctly without checking in the changed field visualization details. However, you should check the field in, to maintain a consistent application definition in the Master Repository.
- Remember, the web server is waiting for a response from your WAM, so the web server could time out if you spend too long stepping through code in debug.

### What You Should Know

- How to check in your WAM and related definitions
- How to run WAMs on the IBM i server from Visual Lansa.

## WAM120 - Using the Menu Bar Weblet

### Objectives

- To demonstrate how to set up the jQuery UI Menubar weblet to provide a fixed application menu.



- The menu will link to four WAMs which handle employee enquiry and update.

To achieve this objective you will complete the following:

[Step 1. Define the Applications Menu](#)

[Step 2. Test the Applications Menu](#)

[Summary](#)

### Before You Begin

You should complete exercises WAM025, WAM030, WAM035 WAM040 and WAM045 which create and then use the common layout **iiilay01**.

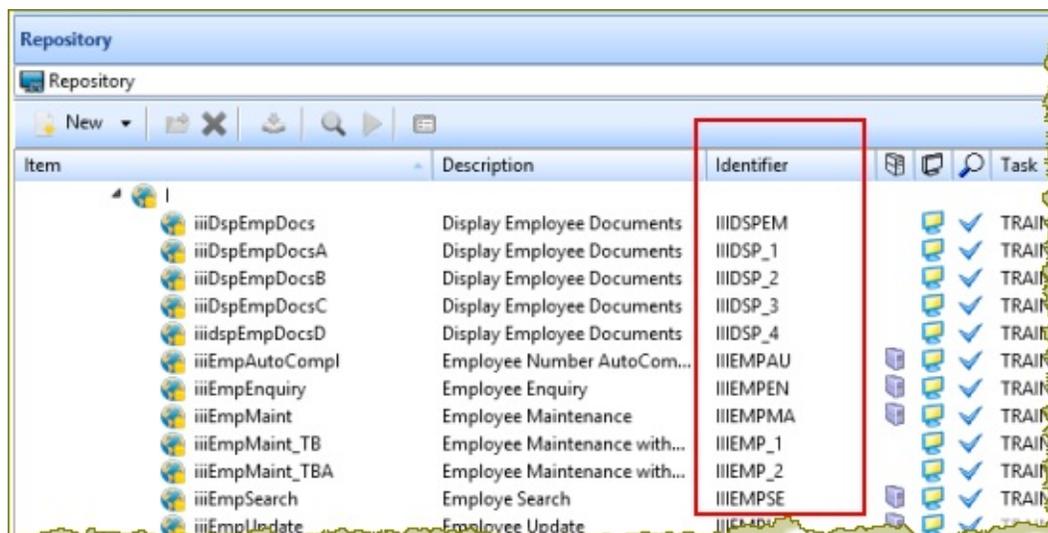
## Step 1. Define the Applications Menu

The menu needs to be setup with WAM name and WebRoutine name for four WAMs. For the first four WAMs using the common layout, the information is as follows. Replace **iii** with your initials.

### New WAMs/Menu Items

WAM Name	Identifier	WebRoutine Name	Menu Caption
iiiEmpEnquiry		begin	Enquiry
iiiEmpUpdate		begin	Update
iiiEmpUpdate_MK2		begin	Update with Dropdowns
iiiDynamSelector		begin	Update with Dropdown using Selector

0. Before you continue with this exercise, display your WAMs on the *Repository* tab and note the **Identifiers** for these WAMs:



Visual LANSA allocates a unique 10 character **Identifier** to each component you create (if Long Names is enabled at partition level). When calling the

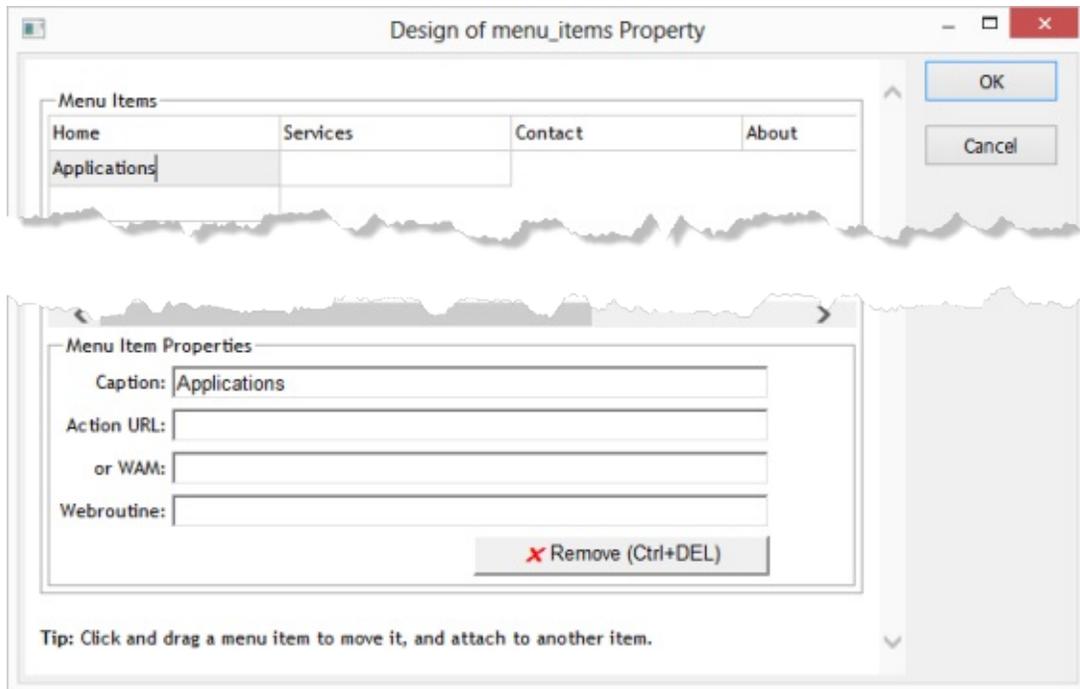
WAM directly in a URL, the Identifier, not the long names must be used.

1. Open layout **iiilay01** in the editor. In exercise WAM025 you added the jQuery UI Menubar weblet to this layout and set up links to a number of LANSAs website pages.
2. Select the menu bar weblet and use the *Ellipsis* button for the *menu\_items* property to open the *Design of...* dialog, which should look like the following:

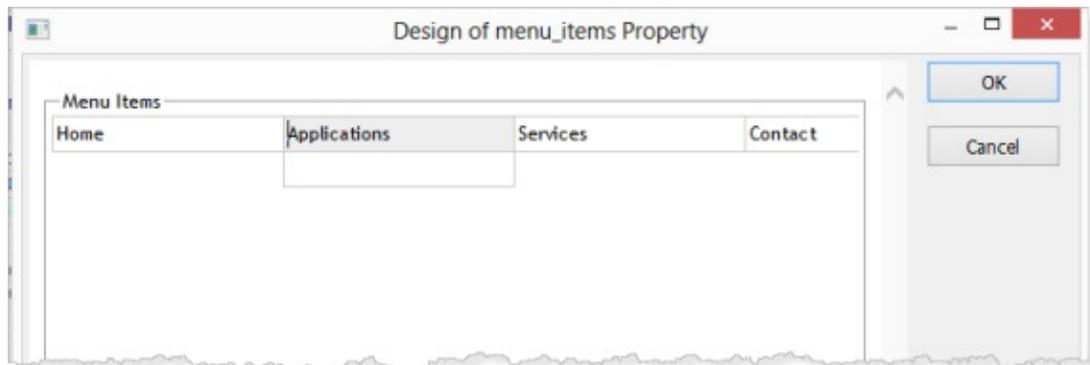


3. In this step you will define an *Applications* top level menu item, underneath menu item *Home* and then drag it into the top level, between *Home* and *Services*.

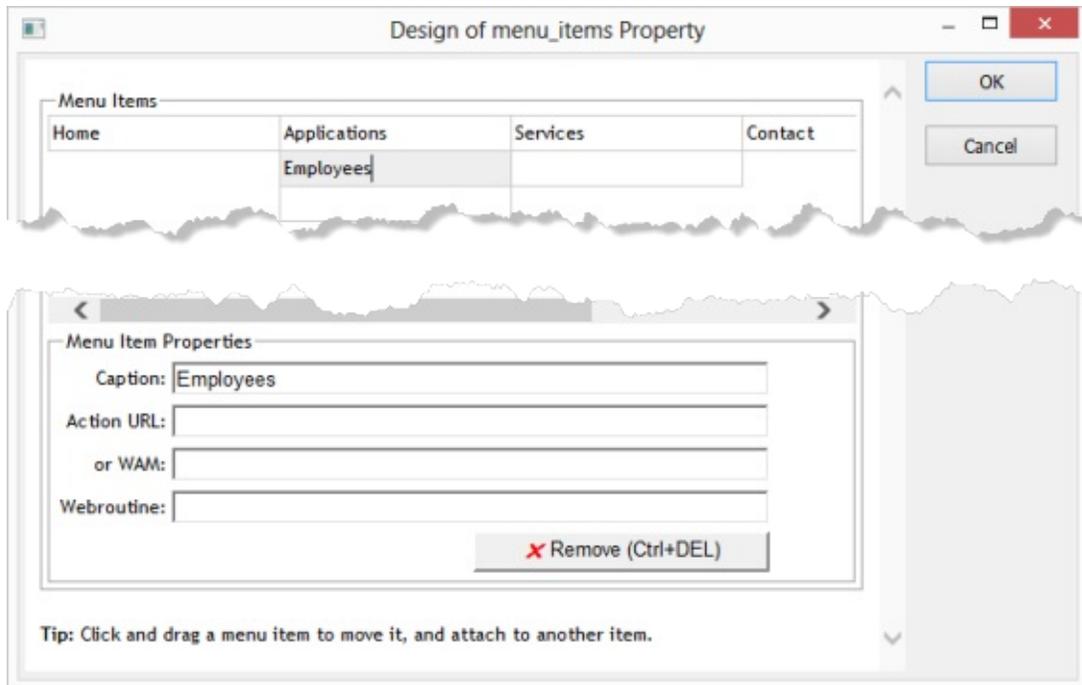
Click in the menu item below *Home* and in the Menu Item Properties, enter *Applications*. The *Design of menu\_items Property* dialog should now look like the following:



- Click on the *Application* menu item and hold down the left mouse button to drag *Applications* into position between *Home* and *Services*. Position the cursor on the left hand side of *Services* when you release the left mouse button. Your design should look like the following:



- Click on the menu item *Applications* to display the sub-menu item underneath it. Enter **Employees** in the Menu Item Properties *Caption*. Your design should look like the following:



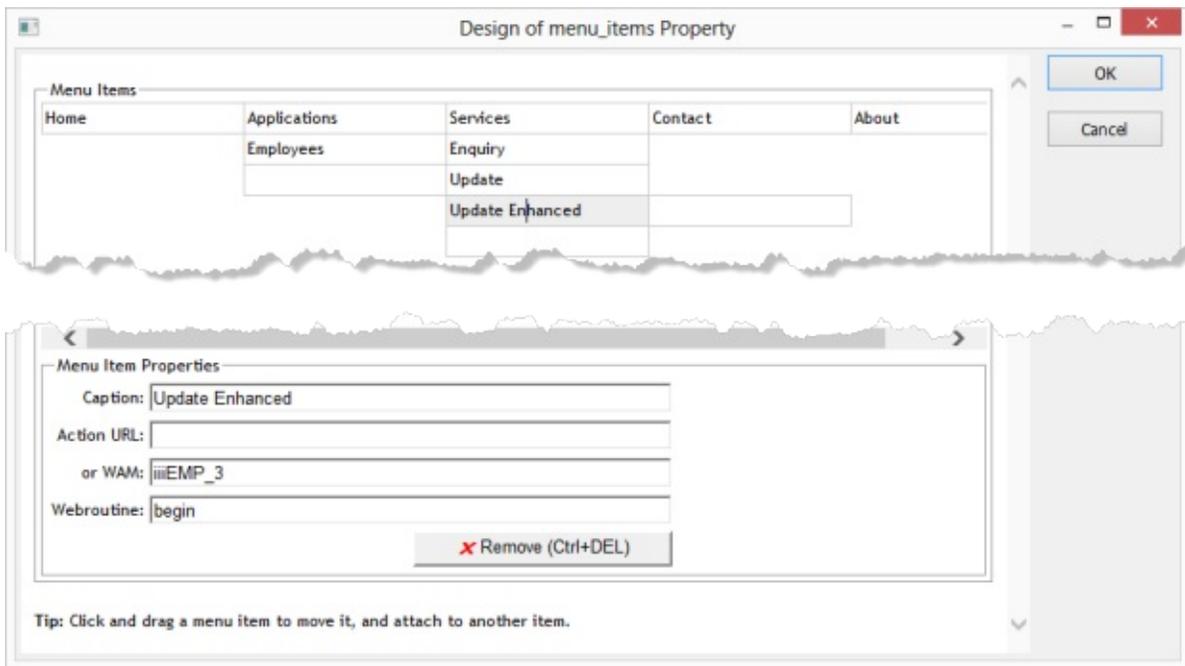
6. Click in the menu item to the right of *Employees*. This means you will be defining a sub-menu for *Employees*, which will be shown during the design step, under *Services*. Define *Enquiry* as:

Property	Value
Caption	Enquiry
WAM	iiiEMPEN
WebRoutine	begin

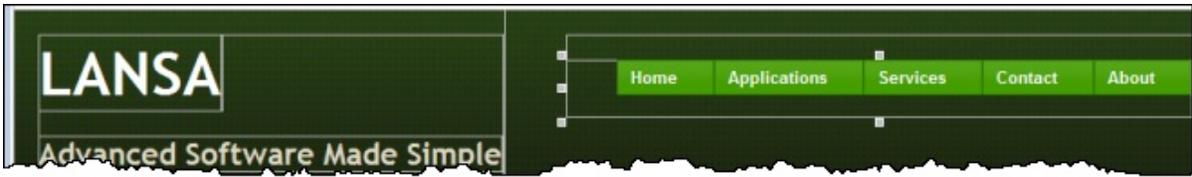
**Note:** Your WAM Identifier may be different to the example shown.  
Your design should look like the following:



- Continue by defining the next three sub-menu items for *Employees*, in the column under *Services*. Refer to the [New WAMs/Menu Items](#) table at the beginning of this step for details. Your dialog should now look like the following:



- Click *OK* to save the changes and close the dialog. Your menu should look like the following:



9. Save your layout.

## Step 2. Test the Applications Menu

1. Open WAM iiiEmpEnquiry in the editor and open the **begin** WebRoutine in the *Design* view. Run the WAM in the browser.
2. Use the Applications menu to run the *Employee Update* WAM:



3. Test all four menu sub-items to ensure that the links have been correctly defined.

## **Summary**

### **Important Observations**

- The menu bar weblet provides the functionality of a menu bar that can invoke other web pages including other web routines.
- The menu bar can be arranged horizontally or vertically and the top level menu items can cause further menus to pop-up as the mouse moves over them.

### **Tips & Techniques**

- The menu items can be defined in the Webroutine design using the menu item designer or they can be supplied at runtime in an RDMLX list.
- The weblet is based on the jQuery UI menu widget and requires jQuery and jQuery UI to operate (the weblet will automatically add the required external resources to the output HTML).

### **What Your Should Know**

- How to define a static menu using the jQuery UI Menubar weblet.

## WAM125 - Define a Dynamic Menu

### Objectives

The jQuery UI Menubar can be defined dynamically using a working list. This exercise demonstrates how a login WAM could establish a custom menu for each group of users.

- A login WAM will handle two users ids ADMIN and USER.
- Only the ADMIN user will have access to the *Applications* menu

To meet these requirements the login WAM requires the following:

- Session Management must be enabled only if the login is valid
- The lists MNUSAVE and MNULIST which define the menu items, must be built with menu sub-items for *Applications*, only for the user ADMIN.
- All WAMs using this menu must be part of the same session group
- WAMs using this menu must transfer to the login WAM if a session is not active
- A working list must be output with fields which support the jQuery UI Menubar weblet.
- Two versions of the menu list are required:
- MNUSAVE will be the persistent version which is saved and restored by all WAMs using this menu.
- MNULIST will be the output version of the list which defines the menubar weblet menu items.
- The menu bar in the common layout **iiilay01** must be set up to use the menu list, MNULIST

To meet these objectives you will complete the following:

[Step 1. Create the Login WAM](#)

[Step 2. Redefine the Menubar in Layout iiilay01](#)

[Step 3. Test your Login WAM](#)

[Step 4. Make Application WAMs part of Session](#)

[Step 5. Test the Applications Menu](#)

[Step 6. Implement Menu for all Employee WAMs \(Optional\)](#)

[Summary](#)

## Step 1. Create the Login WAM

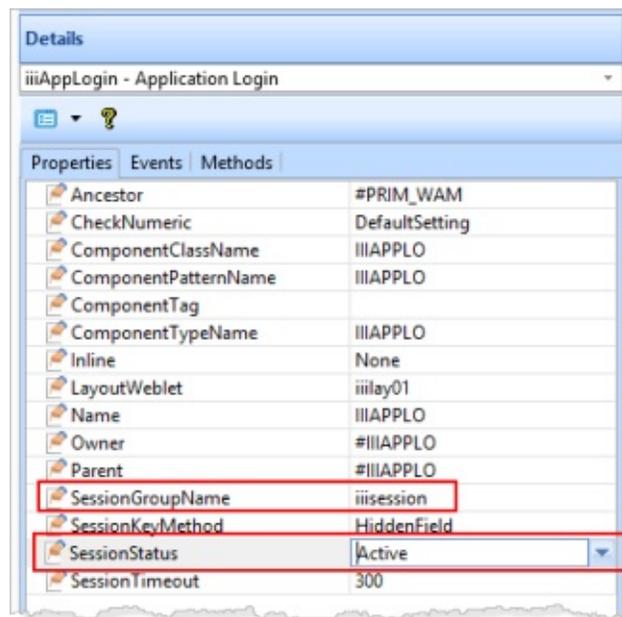
1. Create a new WAM with:

Name: **iiiAppLogin**

Description: **Application Login**

Layout weblet: **iiilay01**

2. Press F7 to display WAM properties. Use the *Details* tab to make *SessionStatus* active and define a *SessionGroupName* of **iiisession**.



3. Add the following definitions of fields and lists which will define the menu

```
Define Field(#mnuitmid) Type(*string) Length(3)
```

```
Define Field(#prtmitid) Type(*string) Length(3)
```

```
Define Field(#mnucapt) Type(*char) Length(30) Input_Atr(LC)
```

```
Define Field(#mnuurl) Type(*string) Length(256)
```

```
Define Field(#mnuwam) Type(*char) Length(9)
```

```
Define Field(#mnuwrnme) Type(*char) Length(30)
```

```
Def_List Name(#mnusave) Fields(#mnuitmid #prtmitid #mnucapt #mnuurl #m
```

```
Def_List Name(#mnulist) Fields(#mnuitmid #prtmitid #mnucapt #mnuurl #mn
```

4. Add the following WEB\_MAPs:

```
Web_Map For(*output) Fields((#mnulist *private))
```

```
Web_Map For(*none) Fields(#mnusave) Options(*persist)
```

```
Web_Map For(*both) Fields((#stdrentry *hidden))
```

Note the way that the map for list MNUSAVE is defined. This is the persistent version of the menu list which will be shared by all WAMs making up this application.

5. Define a WebRoutine **login**, which has an onentry parameter of \*sessionstatus\_none. This WebRoutine will establish a session for valid users and execute the subroutines to build the menu list.

```
Webroutine Name(login) Onentry(*SESSIONSTATUS_NONE)
Web_Map For(*both) Fields(#userid #passwd)
Message Msgtxt('Enter your user id and password')
Endroutine
```

Note that the fields USERID and PASSWD are mapped for \*both. If these fields do not exist in the Repository, define them in your WAM as:

```
Define Field(#USERID) Type(*char) Length(10) Desc("User ID")
Define Field(#PASSWD) Type(*char) Length(10) Desc("Password")
Input_Atr(ND)
```

6. Define a method routine **AddToMenu** which will add each entry to the menu list MNUSAVE. It requires six input parameters which will define the required menu field entries. You will find this information in the *Web Applications Modules Guide / 8.1.18 Menubar (std\_menubar)*. Your code should look like the following:

```
Mthroutine Name(AddToMenu)
Define_Map For(*input) Class(#STD_STRNG) Name(#itmid)
Define_Map For(*input) Class(#STD_STRNG) Name(#pitmid)
Define_Map For(*input) Class(#STD_DESC) Name(#caption)
Define_Map For(*input) Class(#STD_STRNG) Name(#URL)
Define_Map For(*input) Class(#STD_DESCS) Name(#WAM)
Define_Map For(*input) Class(#STD_DESCS) Name(#WRNAME)
#mnuitmid := #itmid
#prtmitmid := #pitmid
#mnucapt := #caption
#mnuurl := #URL
#mnuwam := #WAM
#mnuwrnme := #WRNAME
```

```
Add_Entry To_List(#mnusave)
Endroutine
```

If necessary add field STD\_STRNG to the Repository as a STRING field, length 512.

7. Define a subroutine **buildmenu** which will add the standard menu entries which will be available for all users. This subroutine will be invoked once when the user logs in, so it should begin by clearing MNUSAVE. It should call the method routine AddToMenu to define and add each menu entry. Your code should look like the following:

```
Subroutine Name(buildmenu)
Clr_List Named(#mnusave)
* Home
#com_owner.addtomenu Itmid('1') Pitmid("") Caption('Home') Url('http://www.
* Applications
#com_owner.addtomenu Itmid('2') Pitmid("") Caption('Applications') Url("") Wa
* Support
#com_owner.addtomenu Itmid('3') Pitmid("") Caption('Support') Url('http://ww
* Contact Us
#com_owner.addtomenu Itmid('4') Pitmid("") Caption('Contact Us') Url('http://v
* About
#com_owner.addtomenu Itmid('5') Pitmid("") Caption('About') Url('http://www.
*
Endroutine
```

8. Define a subroutine buildapps which will only be executed for the ADMIN user, which adds entries for the Employees menu and sub-menu items. It should not clear the list MNUSAVE. Your code should look like the following.

As before you must look in the *Repository* for your WAMs and note their Identifiers and replace the values shown in the code below..

```
Subroutine Name(buildapps)
* Employees
#com_owner.addtomenu Itmid('50') Pitmid('2') Caption('Employees') Url("")
Wam("") Wrname("")
* enquiry
#com_owner.addtomenu Itmid('51') Pitmid('50') Caption('Enquiry') Url("")
```

```

Wam(IIIEMPEN) Wrname(BEGIN)
* Update
#com_owner.addtomenu Itmid('52') Pitmid('50') Caption('Update') Url("")
Wam(IIIEMPUP) Wrname(BEGIN)
* Update with Dropdowns
#com_owner.addtomenu Itmid('53') Pitmid('50') Caption('Update with
Dropdowns') Url("") Wam(IIIEMP_3) Wrname(BEGIN)
* Update using Dropdown with Selector
#com_owner.addtomenu Itmid('54') Pitmid('50') Caption('Update using
Dropdown with selector') Url("") Wam(IIIEMP_4) Wrname(BEGIN)
Endroutine

```

**Note:** All menu entries must have a unique id. Sub-menu items must also have the correct parent id.

- Define a new WebRoutine **welcome**. The **login** WebRoutine will transfer to this after a valid login. Welcome will clear and populate the output menu list MNULIST and display "welcome" messages. Your code should look like the following:

```

Webroutine Name(welcome)
Web_Map For(*input) Fields(#userid)
Clr_List Named(#mnulist)
Selectlist Named(#mnusave)
Add_Entry To_List(#mnulist)
Endselect
Message Msgtxt('Welcome ' + #USERID) Type(*STATUS)
Message Msgtxt('You are logged into the Personnel Applications System') Typ
Endroutine

```

Note that the USERID is mapped into this WebRoutine so that it's available in the welcome message.

- In this step you will complete the **login** WebRoutine.

The login button will be set up to return STDREENTRY with a value of L.

A CASE loop for field USERID will check password, make the session status active, and execute one or both build menu subroutines. It will then transfer to the **welcome** WebRoutine. Your completed **login** WebRoutine should look like the following:

```

Webroutine Name(login) Onentry(*SESSIONSTATUS_NONE)
Web_Map For(*both) Fields(#userid #passwd)
Message Msgtxt('Enter your user id and password')
If (#stdrentry = L)
Case (#userid)
When (= ADMIN)
If (#passwd = '14MIN')
#com_owner.sessionstatus := active
Execute Subroutine(buildmenu)
Execute Subroutine(buildapps)
Transfer Toroutine(welcome)
Else
Message Msgtxt('Password incorrect for this user')
Endif
When (= USER)
If (#passwd = 'US5R')
#com_owner.sessionstatus := active
Execute Subroutine(buildmenu)
Transfer Toroutine(welcome)
Else
Message Msgtxt('Password incorrect for this user')
Endif
Endcase
Endif
Endroutine

```

11. Compile your WAM.

12. Open the **login** WebRoutine in the *Design* view.

- a. Add a row to the bottom of the fields table
- b. Add a push button with image to the bottom right hand cell
- c. Set up the push button as follows:

Property	Value
caption	Log In
left_relative_image	icons/normal/16/user_16.png
on_click_wrname	Login

submitExtraFields

Field Name: STDREENTRY

Literal Value: L

13. Save your changes.

## Step 2. Redefine the Menubar in Layout **iiilay01**

1. Open the layout **iiilay01** in the editor.
2. Select the menu bar weblet.
3. Open the *Design of ...* dialog for the *menu\_items* property and delete all the hard coded menu items. Click *OK* to save the changes.
4. Define the *listname* property as **MNULIST**.
5. Save your changes.

### Step 3. Test your Login WAM

1. Make a note of the userid and password values which have been hard coded. Execute the login WebRoutine and login as ADMIN. The menu bar should look like the following:



2. If you are experiencing problems, try using debug to investigate.
3. Run the **Login** WebRoutine again and log in as USER. The menu bar should display the *Applications* top level menu item, but no sub-menu items will be shown.

## Step 4. Make Application WAMs part of Session

In this step you will enable session management in WAM `iiiEmpEnquiry` and make it share the session group `IISESSION`. The WAM will need to map the saved and output menu lists in the same way as already implemented in the log in WAM `iiiAppLogin`.

1. Open WAM `iiiEmpEnquiry` in the editor. Press F7 to display the WAM properties and make `SessionStatus` **active** and `SessionGroupName`, **IISESSION**.
2. Copy and paste the following field, list and `web_maps` statements from `iiiAppLogin`. In a real application the menu fields would be defined in the *Repository*.

```
Define Field(#mnuitmid) Type(*string) Length(3)
Define Field(#prtmitid) Type(*string) Length(3)
Define Field(#mnucapt) Type(*char) Length(30) Input_Atr(LC)
Define Field(#mnuurl) Type(*string) Length(256)
Define Field(#mnuwam) Type(*char) Length(9)
Define Field(#mnuwrnme) Type(*char) Length(30)
Def_List Name(#mnusave) Fields(#mnuitmid #prtmitid #mnucapt #mnuurl #m
Def_List Name(#mnulist) Fields(#mnuitmid #prtmitid #mnucapt #mnuurl #mn
Web_Map For(*output) Fields((#mnulist *private))
Web_Map For(*none) Fields(#mnusave) Options(*persist)
```

3. Add an event handling routine for `SessionInvalid`. This will transfer to WAM **iiiAppLogin** and WebRoutine **login** if any WebRoutine is invoked without a session being established or after a session has timed out. Change **iii** to your initials.

```
EVTROUTINE HANDLING(#COM_OWNER.SessionInvalid) OPTIONS(*N
TRANSFER TOROUTINE(#iiiAppLogin.login)
ENDROUTINE
```

4. Change the initial WebRoutine in **iiiEmpEnquiry**, in this case it is WebRoutine **begin**, to clear list `MNULIST` and populate it from `MNUSAVE`. Your WebRoutine **begin** code should now look like the following. Changes are shown in red.

```
Webroutine Name(begin) Desc('Select Employee')
```

```
Web_Map For(*output) Fields(#empno)
Clr_List Named(#mnulist)
Selectlist Named(#mnusave)
Add_Entry To_List(#mnulist)
Endselect
Endroutine
```

Remember list MNUSAVE is automatically restored when the WAM runs, because session management is active and the list is mapped as persistent data. Both field values and lists may be defined as persistent data.

5. Compile iiiEmpEnquiry.

## Step 5. Test the Applications Menu

1. Execute the **login** WebRoutine in `iiiAppLogin` in the browser. Log in as user ADMIN.
2. From the menu bar select *Applications / Employees / Enquiry* to execute WebRoutine **begin** in WAM **iiiEmpEnquiry**. With the *Enquiry* WAM running check that the *Applications* menu is shown. Close the browser.
3. Execute the **begin** WebRoutine in **iiiEmpEnquiry** in the browser. Your application should transfer to the log in WAM so that a session can be established. The transfer is handled via the event handling routine for `SessionInvalid`.

## **Step 6. Implement Menu for all Employee WAMs (Optional)**

1. If the changes made in [Step 4. Make Application WAMs part of Session](#) are applied to WAMs iiiEmpUpdate, iiiEmpUpdate\_MK2 and iiiDynamSelector then all four Employee applications may be run from the menu and will display the correct menu.

## Summary

### Important Observations

- In a real application you could define login authority in a database file with a small application to manage them.

### Tips & Techniques

- The menu bar has a *submit\_selected\_to* property, which will send the *menu id* in the defined field to the WebRoutine which is invoked. This could enable the WebRoutine to behave in a specific way when it is first invoked from the menu.

### What You Should Know

- How to implement the jQuery UI Menubar using a working list.

## **WAM130 - Output a Web Page to a File**

### **Objectives**

Running a WAM using the X\_RUN command, enables the WebRoutine output to be saved to a file. This enables permanent (instead of dynamic) web pages to be created from your application. This may be an advantage when seeking to optimize search engines searches over your public web site. It can also be used as a method of saving results which will be fixed for a period of time (for example monthly statistics) rather than repeatedly running a WAM to calculate the same set of results for each user enquiry.

Output to a file from a WebRoutine is supported for Windows, IBM i and Linux servers but you need to be aware of differences with the X\_RUN command parameters used. See a [Saving a WAM's Output to a File](#) for more details.

Since the WebRoutine output goes directly to an HTML file, it can only contain output generated by the WebRoutine.

To demonstrate WAM output to a file on Windows and IBM i (if available) you will complete the following:

[Step 1. Output Employee Enquiry to a File](#)

[Step 2. Run WAM to output to a file in Windows](#)

[Step 3. Run WAM to output a file on IBM i](#)

[Summary](#)

### **Before You Begin**

Complete the introductory exercises, WAM005, WAM010, WAM015 and WAM020 before starting this exercise.

## Step 1. Output Employee Enquiry to a File

This exercise uses a simple enquiry WAM. When running in the browser, an employee number is requested via WebRoutine **begin**. A second WebRoutine **details** is invoked which displays employee data.

Using X\_RUN the **details** WebRoutine is invoked, passing employee number as a parameter so that the web page containing data for the requested employee can be written to a file.

To execute a WAM, the parameters required by the X\_RUN command for Windows are as follows:

### Argument Value

PROC	*WAMSP is the value to activate the "output to file" function
WMOD	WAM Name
WRTN	WebRoutine Name
WAML	Mark up language, e.g. LANSA:HTML
PART	Partition
LANG	Language, e.g ENG
USER	User. Optional for some platforms
WASP	Output file path where the output will be saved. Format depends on platform. e.g. for Windows c:\temp\wam_output.html

The format of the X\_RUN command for Windows is:

```
X_RUN PROC=*WAMSP WMOD=IIIEmpEnqToFile WRTN=DETAILS . . .
```

The X\_RUN command does not have a parameter to input a field and value. X\_RUN does have a user defined parameter UDEF, which is a 256 long character field. To use the UDEF parameter, a simple modification to the WebRoutine is required to retrieve the UDEF value using the Built-in

Function GET\_SESSION\_VALUE. If the value returned by the BIF is non blank, the **details** WebRoutine can retrieve employee data using the session value as employee number.

1. Create a new Employee Enquiry WAM and complete its code by copying the following code.

*Name:* **iiiEmpEnqToFile**

*Description:* **Employee Enquiry to File**

*Layout Template:* **iiilay01**

```
Group_By Name(#empgroup) Fields((#SURNAME *OUT) (#GIVENAME *C
Webroutine Name(begin) Desc('Select Employee')
Web_Map For(*output) Fields(#empno)
Endroutine
Webroutine Name(DETAILS) Desc('Employee Details')
Web_Map For(*BOTH) Fields((#EMPNO *OUTPUT))
Web_Map For(*OUTPUT) Fields(#empgroup)
Fetch Fields(#empgroup) From_File(PSLMST) With_Key(#EMPNO) Val_Err
If_Status Is_Not(*OKAY)
Message Msgtxt('Employee not found')
Endif
Endroutine
```

2. Add logic to the details WebRoutine to retrieve session value. If non blank, use as employee number.

Add the following code in WebRoutine **Details**, immediately **before** the FETCH command.

```
Define Field(#retcode) Type(*char) Length(2)
Use Builtin(get_session_value) With_Args(UDEF) To_Get(#STD_QSEL #retc
If (#std_qsel *NE *blanks)
#empno := #std_QSEL.trim
Endif
```

3. Compile your WAM.
4. Open the **begin** WebRoutine in the design view.
  - a. Add a right hand column to the table containing employee number.
  - b. Drag and drop a push button weblet into the new column.

- c. Change the *caption* to **Select**.
  - d. Set the push button *on\_click\_wrname* property to invoke the **Details** WebRoutine.
  - e. The *submitExtraFields* property does not need to be specified.
  - f. Remove the place holder characters.
  - g. Save your changes.
5. Run the **begin** WebRoutine in the browser and ensure that the **details** WebRoutine executes normally when invoked from the **begin** web page. Enter an employee number such as A0090, A0070 or A1001.

## Step 2. Run WAM to output to a file in Windows

This step will run the **details** WebRoutine in Windows using the X\_RUN command and pass in the employee number using the UDEF, a user defined run time parameter.

The X\_RUN.EXE program is in the Visual LANSA \execute folder, for example:

```
C:\Program Files\LANSA_D12\X_WIN95\X_LANSA\execute
```

You could add this path to your Windows Path environment variable, in which case the X\_RUN program will be found when it is run from a simple batch file. For this exercise you will create a DOS batch file using Notepad and set up the required path.

1. Open Notepad and add the following code:

```
cd\  
cd program files\lansa\x_win95\x_lansa\execute  
x_run PROC=*WAMSP WMOD=iiiEMP_4 WRTN=DETAILS WAML=LAN
```

**Important:** With your WAM open in the editor, select the *Repository Details* tab to find its *Identifier*. This must be used in the X\_RUN (that is, replace WMOD=IIIEMP\_4 in this code).

If necessary change the change directory command (cd) to reflect your Visual LANSA installed path.

Review the X\_RUN command carefully and correct it for your Visual Lanssa path, partition name, user and output HTML file name.

The specific X\_RUN parameters you may need to change are:

WMOD – WAM Identifier

PART – partition name

USER – user name (use your Visual LANSA profile name)

WASP – The output path and file name. Use your initials

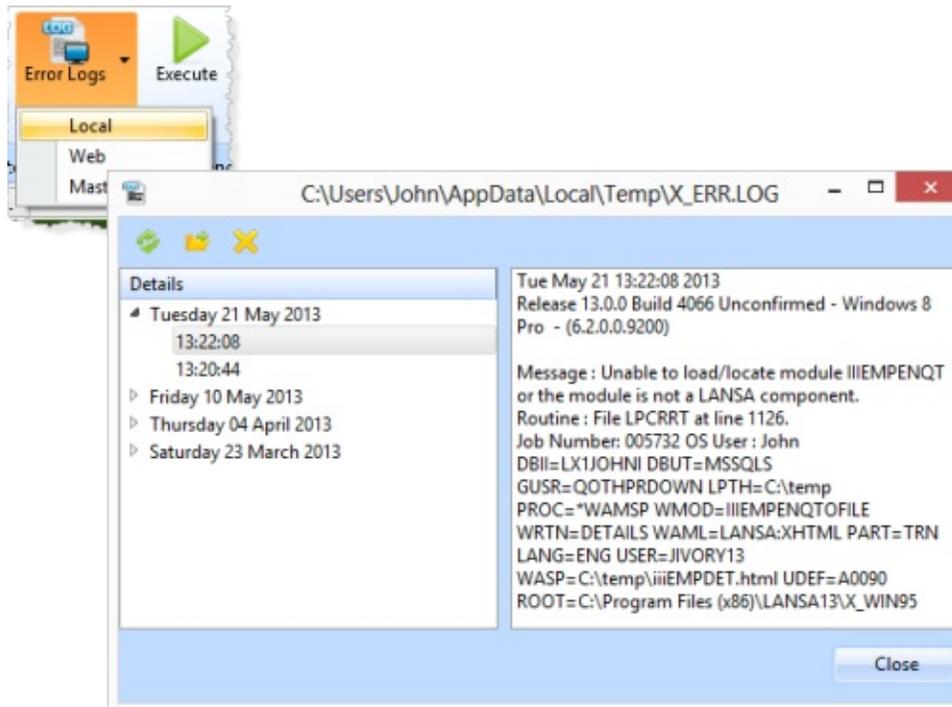
UDEF – The employee number

Create a c:\temp folder if necessary.

2. Use the *Save as Type: All* to save the file as iiiWAMOUT.bat in folder c:\temp

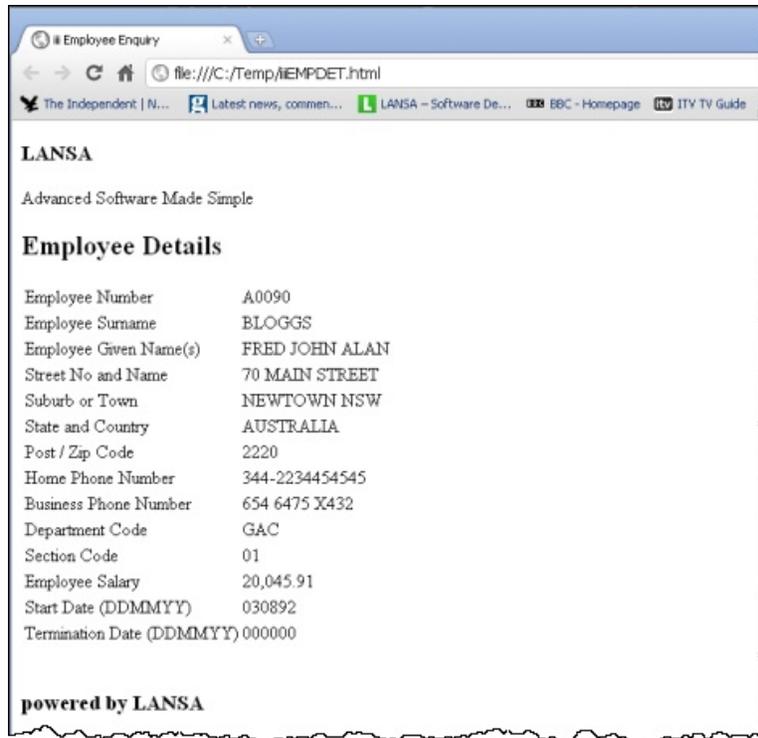
3. Navigate to c:\temp with Windows Explorer and double click on iiiWAMOUT.bat to run it. The DOS command prompt should briefly open and close.
4. Check folder c:\temp for the output file iiiEMPDET.html.
5. If your run does not output the file iiiEMPDET.html, check the batch file carefully for errors and try again.

When running your WAM locally, you can also check for errors in the *Visual Lansa Error Log*.



See also the *Web Runtime Error Log* from the *Error Logs* menu.

6. When your WAM has created the output file, double click on it to open it directly in your PC's default browser. It should look like the following:



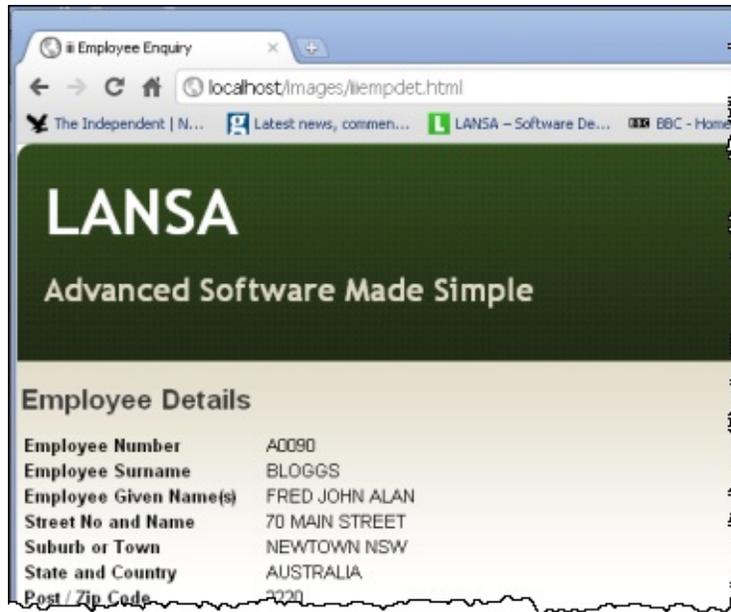
No cascading style sheets have been applied.

7. Copy the output file iiiEMPDET.html to your Visual LANSA web server\images path, such as:

C:\Program Files\LANSA\WebServer\Images

8. Open the file by putting the following URL into your browser address bar:  
<http://localhost/images/iiiempdet.html>

Your web page should now be displayed correctly formatted, because the cascading style sheets have been applied. See below:



### Step 3. Run WAM to output a file on IBM i

1. Check in and compile WAM iiiEmpEnqToFile. Its layout will be included in the check in list of objects, automatically. Also check in the common layout iiilay01, if it has not been checked in before. Review the *Check In* tab to ensure the Check In was successful
2. Start an emulator session. Log on with your Visual LANSA user id and password. Call the command processor with

#### **CALL QCMD**

Display a full screen by pressing F11.

3. Enter the following command:

```
LANSA REQUEST(X_RUN) PARM01('PROC=*WAMSP')  
PARM02('WMOD=IIIEMP_4') PARM03('WRTN=DETAILS')  
PARM04('WAML=LANSA.XHTML') PARM05('PART=TRN')  
PARM06('LANG=ENG') PARM07('/WASP=/TMP/IIIEMPDET.HTML')  
PARM08('UDEF=A0090')
```

**Note:** You can copy this command string from *WAM Tutorials* in the online guide and use the *Edit/Paste* menu option in Client Access emulator to copy the command into the 5250 screen. Then press F4 to prompt the LANSA command.

Correct the call command for your WAM Identifier name, partition name and user name. Check that your IBM i has a /tmp folder in the IFS or use a suitable alternative folder.

Press Enter to run the WAM.

4. Use WRKLNK to view the /tmp folder and find your output HTML file.

**Note:** Your call command should have run successfully because your library list was correct, so that the X\_RUN program was found. You logged on a LANSA developer and therefore had the correct library list.

If you were creating a CL program to run this job, you would need to establish the correct library list. For example:

```
ADDLIB <PGMLIB>  
LANSA REQUEST(X_RUN) PARM01('PROC=*WAMSP')  
PARM02('WMOD=IIIEMP_4') PARM03('WRTN=DETAILS')  
PARM04('WAML=LANSA.XHTML') PARM05('PART=TRN')
```

```
PARM06('LANG=ENG') PARM07('WASP=/TMP/IIIEMPDET.HTML')  
PARM08('UDEF=A0090')
```

where: <PGMLIB> is your LANSAs program library.

You could also consider error handling in the modified **details** WebRoutine. What to do if the UDEF variable is non-blank but the FETCH employee record is unsuccessful?

Once again, if you have access to the /tmp folder from Windows explorer the iiiempdet.html can be opened in the browser, but it will be unformatted.

5. Copy the output HTML file to the appropriate IBM i web server /images folder. For example:

```
CPY OBJ('/tmp/iiiempdet.html')  
TODIR('/lansa_<PGMLIB>/webserver/www/htdocs  
images') REPLACE(*YES)
```

where <PGMLIB> is your LANSAs program library.

Check that the TODIR parameter is correct for you installation of LANSAs.

6. Put the following URL into your browser address bar:

```
http://10.44.10.238/images/iiiempdet.html
```

**where:** 10.44.10.238 is your IBM i IP Address (or use its server name). Your saved web page should now be displayed correctly formatted and the required style sheets (CSS) will have been included.

## Summary

### Important Observations

- For further details on using this facility see the [Saving a WAM's Output to a File](#).

### Tips & Techniques

- The user defined parameter accepted by X\_RUN is a 256 long character string. You could input a more complex key (DEPARTMENT and SECTION to identify a section record for example) by concatenating them into UDEF. The WebRoutine would then split the UDEF variable into its known parts.

### What You Should Know

- How to implement the *Save WAM output to a file* feature.

## WAM135 - Using the Google Static Maps API

### Objective

This exercise provides a very simple example of using address information to display a street map using the Google Static Map API. The Google Maps facility can be used in many more sophisticated ways. This example is intended to illustrate that provided you have good address data, a static map can quite easily be displayed.

The information to build this example was taken from:

[http://code.google.com/apis/maps/documentation/staticmaps/#quick\\_example](http://code.google.com/apis/maps/documentation/staticmaps/#quick_example)

See the following URL for full documentation:

<http://code.google.com/apis/maps/documentation/staticmaps/>

The supplied example shows how the following <img> tag will display a street map around the Brooklyn Bridge, New York.

```
<img src=http://maps.googleapis.com/maps/api/staticmap?
center=Brooklyn+Bridge,New+York,NY&zoom=14&size=512x512&maptype
&markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:;
&markers=color:red%7Ccolor:red%7Clabel:C%7C40.718217,-73.998284&sei
```

A simple employee enquiry will display a static street map for the employee's address.

To meet this objective you will complete the following:

[Step 1. Create an Employee Enquiry WAM](#)

[Step 2. Add logic to set up URL to Google Map Service](#)

[Summary](#)

### Before You Begin

Complete the introductory exercises, WAM005, WAM010, WAM015 and WAM020.

## Step 1. Create an Employee Enquiry WAM

1. Create a new WAM with:

*Name:* **iiiEmpEnqMap**

*Description:* **Employee Enquiry with Map**

*Layout Template:* **iiilay01**

Create the WAM by copying the following code into the default WAM source:

```
Group_By Name(#empgroup) Fields((#SURNAME *OUT) (#GIVENAME *C
Define #ADDRESS Type(*string) Length(1000)
Webroutine Name(begin) Desc('Select Employee')
Web_Map For(*output) Fields(#empno)
Endroutine
Webroutine Name(DETAILS) Desc('Employee Details')
Web_Map For(*BOTH) Fields((#EMPNO *OUTPUT)
(#ADDRESS *hidden))
Web_Map For(*OUTPUT) Fields(#empgroup)
Fetch Fields(#empgroup) From_File(PSLMST) With_Key(#EMPNO) Val_Err
If_Status Is_Not(*OKAY)
Message Msgtxt('Employee not found')
Transfer Toroutine(begin)
Endif
Endroutine
```

2. Compile the WAM.
3. Add a push button to the begin web page.
  - a. Open the **begin** WebRoutine in the *Design* view.
  - b. Add a column to the fields table.
  - c. Add a push button in the new column.
  - d. Set up the button with a caption of **Details** and an *on\_click\_wrname* of **details**.
  - e. Remove the \* placeholder characters.
  - f. Save your changes.
4. Open the **Details** WebRoutine in the *Design* view.

- a. Add a column to the fields table
- b. With the top row in the new column selected, set its *rowspan* property to **14**.
- c. Click inside the new column and use the context menu to *Insert HTML / Div*. Your design should look like the following:

The screenshot shows a web form titled "Employee Details" with the following fields and values:

Employee Number	ABCDE
Employee Surname	ABCDEFGHJKLMNOPQRST
Employee Given Name(s)	ABCDEFGHJKLMNOPQRST
Street No and Name	aAbBcCdDeEfGhHiIjJkKlLm
Suburb or Town	aAbBcCdDeEfGhHiIjJkKlLm
State and Country	aAbBcCdDeEfGhHiIjJkKlLm
Post / Zip Code	123456
Home Phone Number	ABCDEFGHJKLMNOP
Business Phone Number	ABCDEFGHJKLMNOP
Department Code	Value DEPARTMENT
Section Code	AB
Employee Salary	123,456,789.12
Start Date (DDMMYY)	12/34/56
Termination Date (DDMMYY)	12/34/56

- d. With the DIV still selected, select the *Details* tab. Expand its *Style* property and set the following properties:

Property	Value
Height	340px
Margin-left	50px
Width	340px

Your design should look like the following:

### Employee Details

Employee Number	ABCDE	
Employee Surname	ABCDEFGHJKLMNOPQRST	
Employee Given Name(s)	ABCDEFGHJKLMNOPQRST	
Street No and Name	aAbBcCdDeEffGhHiIjJkKlLm	
Suburb or Town	aAbBcCdDeEffGhHiIjJkKlLm	
State and Country	aAbBcCdDeEffGhHiIjJkKlLm	
Post / Zip Code	123456	
Home Phone Number	ABCDEFGHJKLMNOP	
Business Phone Number	ABCDEFGHJKLMNOP	
Department Code	Value DEPTMENT	
Section Code	AB	
Employee Salary	123,456,789.12	
Start Date (DDMMYY)	12/34/56	
Termination Date (DDMMYY)	12/34/56	

- e. Delete the \* place holder characters from this column.
- f. Save your changes.
- g. Drop a *clickable image* weblet into the DIV.
- h. Select the *clickable image* and set its *absolute\_image\_path* to **#ADDRESS**.  
Enter #ADDRESS in the *XPath Expression* window.
- i. Save your changes.

## Step 2. Add logic to set up URL to Google Map Service

In this step you will extend the **Details** WebRoutine to use the employee address fields to set up the URL string which will be output as the #ADDRESS field.

Review the example Google URL provided in *Objectives*, and note the Brooklyn Bridge address values in that string.

An employee address such as:

70 MAIN STREET (field ADDRESS1)

NEWTOWN NSW (field ADDRESS2)

AUSTRALIA (field ADDRESS3)

Must be used to produce a string inside the Google URL such as:

MAIN+STREET,NEWTOWN,NSW,AUSTRALIA

Your code needs to do the following:

- Remove house number
- Remove leading spaces
- Add + between words in an address line
- Add a comma between elements in field ADDRESS2

1. Define two work fields at WAM level (below the Begin\_com):

```
Define Field(#addr1) Reffld(#address1)
```

```
Define Field(#addr2) Reffld(#address2)
```

2. In the **Details** WebRoutine add the following logic, immediately after the FETCH statement.

a. Remove numbers and first space character from ADDRESS1

```
#addr1 := #address1.removecharacters( "1234567890" ).remove( " " )
```

b. Replace space character (between words) with a +

```
#addr1 := #addr1.replace( " ", "+" )
```

c. Replace space character in ADDRESS2 with comma.

```
#addr2 := #address2.replace( " ", "," )
```

d. Define the field ADDRESS as the complete Google URL

```
#ADDRESS := 'http://maps.googleapis.com/maps/api/staticmap?center=' +  
#addr1 + ',' + #addr2 + ',' + #address3 +  
'&zoom=14&size=340x340&maptype=roadmap&markers=color:blue%7C' +  
#addr1 + ',' + #addr2 + ',' + #address3 + '&sensor=false'
```

Note that the size value in the URL has been changed to 340x340.

All other values use the supplied example values for example, for color. These could all be adjusted by reference to the Google web site for this service.

3. Compile and test your WAM. Your details web page should now look like the following:

Employee Details	
Employee Number	A0090
Employee Surname	BLOGGS
Employee Given Name(s)	FRED JOHN ALAN
Street No and Name	70 MAIN STREET
Suburb or Town	NEWTOWN NSW
State and Country	AUSTRALIA
Post / Zip Code	2220
Home Phone Number	344-2234454545
Business Phone Number	654 6475 X432
Department Code	
Section Code	
Employee Salary	20,045.91
Start Date (DDMMYY)	030892
Termination Date (DDMMYY)	000000



## **Summary**

### **Important Information**

- You should read the terms and conditions for this service.

### **Tips & Techniques**

- There are many services like this example which can be used to enhance your web applications.

### **What You Should Know**

- How to implement the Google Static Map service.

## **Appendix A. XSL and XML Conformance**

LANSA for the Web WAMs conform to the following W3C XML standards:

- XML 1.0 standard: <http://www.w3.org/TR/REC-xml/>
- Namespaces in XML: <http://www.w3.org/TR/REC-xml-names/>
- XSLT 1.0 standard: <http://www.w3.org/TR/xslt>
- XPath 1.0 standard: <http://www.w3.org/TR/xpath>

## **Appendix B. WAM XML Structure**

The XML document produced from a WEBROUTINE invocation at runtime has a standard format. All fields, lists and other output from a WEBROUTINE are defined in the XML document. This XML document is used as input to the transformation using the WEBROUTINE XSL stylesheets to produce the final presentation output. The format of the XML document must be well known and standard to enable correct transformation into different and distinct presentation formats.

The XML document is divided into the following sections:

### **Context Section**

The context section in the XML document contains contextual information about the WEBROUTINE. Items such as WAM name, WEBROUTINE name, WEBROUTINE title are available here.

### **Options Section**

The options section contains various options that may be modified for a WEBROUTINE that may determine whether particular validation or presentation functionality is enabled or not.

### **Messages Section**

The messages section contains messages output using the MESSAGE RDML command at runtime.

### **Fields Section**

The fields section contains fields that appear as outgoing fields in WEB\_MAP statements in the WEBROUTINE. At runtime, the outgoing field values are added to this section to be transformed into the presentation output. Also captions, descriptions and headings of those fields are added to this section both at runtime and design time.

### **Lists Section**

The lists section contains lists that appear as outgoing lists in WEB\_MAP statements in the WEBROUTINE. Headings for each of the field columns and field values for each of the list rows for each of the lists are added to this section at runtime. Headings, but not actual runtime row values, are also available in this section at design time. Generated lists have an attribute inline="true" if they are inlined. JSON lists are sent as CDATA sections. JSON lists are normally consumed by JavaScript.

## XML Document Example

The following is an example of the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Web application : SAMPLEWAM    Test
      Webroutine    : EmployeeEntry  Add an Employee
      Timestamp     : 2011-08-26T16:45:00 -->
<lxml:data xmlns:lxml="http://www.lansa.com/2002/XML/Runtime-Data">
  <lxml:context>
    <lxml:user-id>PCXUSER</lxml:user-id>
    <lxml:webapplication>SAMPLEWAM</lxml:webapplication>
    <lxml:webapplication-title>Test</lxml:webapplication-title>
    <lxml:webroutine>EmployeeEntry</lxml:webroutine>
    <lxml:webroutine-title>Add an Employee</lxml:webroutine-title>
    <lxml:service-name>SAMPLEWAM_EmployeeEntry</lxml:service-
name>
    <lxml:partition>DEM</lxml:partition>
    <lxml:language iso-lang="en">ENG</lxml:language>
    <lxml:images-path>/IMAGES</lxml:images-path>
    <lxml:action-request>/CGI-BIN/lansaweb</lxml:action-request>
  </lxml:context>
  <lxml:options>
    <lxml:option name="DBCS">>false</lxml:option>
    <lxml:option name="align-right">>true</lxml:option>
    <lxml:option name="check-numeric">>true</lxml:option>
    <lxml:option name="debug" />
    <lxml:option name="trace" />
    <lxml:option name="task" />
  </lxml:options>
  <lxml:external-resources>
    <lxml:script name="XWJQC" charset="utf-8"
location="header">jquery/1.9.1/jquery.min.js</lxml:script>
    <lxml:script name="XWJQUI" charset="iso-8859-1"
location="header">jquery-ui/1.10.3/js/jquery-ui.all.min.js</lxml:script>
    <lxml:script name="XWJ001" charset="utf-8"
location="header">script/std_jqueryui.min.js</lxml:script>
    <lxml:script name="XWJ003" charset="utf-8"
location="header">script/std_json.min.js</lxml:script>
```

```

    <lxml:style name="XWC001" charset="utf-8"
location="header">style/jquery/std_jqueryui.min.css</lxml:style>
    <lxml:style name="XWT01J" charset="iso-8859-1"
location="header">jquery-ui/1.10.3/css/redmond/jquery-
ui.all.min.css</lxml:style>
    <lxml:style name="XWT01L101" charset="utf-8"
location="header">style/jquery/redmond/std_themelet1_style1.min.css</lxml:
</lxml:external-resources>
<lxml:messages />
<lxml:fields>
    <lxml:field name="SURNAME">
        <lxml:caption>
            <lxml:label>Surname.....</lxml:label>
            <lxml:description>Employee Surname</lxml:description>
            <lxml:heading-1>Surname</lxml:heading-1>
            <lxml:heading-2 />
            <lxml:heading-3 />
        </lxml:caption>
        <lxml:value />
    </lxml:field>
    <lxml:field name="GIVENAME">
        <lxml:caption>
            <lxml:label>Given names....</lxml:label>
            <lxml:description>Employee Given Name(s)</lxml:description>
            <lxml:heading-1>Given name(s)</lxml:heading-1>
            <lxml:heading-2 />
            <lxml:heading-3 />
        </lxml:caption>
        <lxml:value />
    </lxml:field>
    <lxml:field name="EMPNO">
        <lxml:caption>
            <lxml:label>Employee no....</lxml:label>
            <lxml:description>Employee Number</lxml:description>
            <lxml:heading-1> Employ</lxml:heading-1>
            <lxml:heading-2> Number</lxml:heading-2>
            <lxml:heading-3 />
        </lxml:caption>
        <lxml:value />

```

```
</lxml:field>
<lxml:field name="ADDRESS1">
  <lxml:caption>
    <lxml:label>Address 1.....</lxml:label>
    <lxml:description>Street No and Name</lxml:description>
    <lxml:heading-1>Address line 1</lxml:heading-1>
    <lxml:heading-2 />
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="ADDRESS2">
  <lxml:caption>
    <lxml:label>Address 2.....</lxml:label>
    <lxml:description>Suburb or Town</lxml:description>
    <lxml:heading-1>Address line 2</lxml:heading-1>
    <lxml:heading-2 />
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="ADDRESS3">
  <lxml:caption>
    <lxml:label>Country</lxml:label>
    <lxml:description>State and Country</lxml:description>
    <lxml:heading-1>Country</lxml:heading-1>
    <lxml:heading-2 />
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="POSTCODE">
  <lxml:caption>
    <lxml:label>Post/zip code..</lxml:label>
    <lxml:description>Post / Zip Code</lxml:description>
    <lxml:heading-1>Post/zip</lxml:heading-1>
    <lxml:heading-2>Code</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
```

```
<lxml:value />
</lxml:field>
<lxml:field name="PHONEHME">
  <lxml:caption>
    <lxml:label>Home phone.....</lxml:label>
    <lxml:description>Home Phone Number</lxml:description>
    <lxml:heading-1>Home phone</lxml:heading-1>
    <lxml:heading-2>Number</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="PHONEBUS">
  <lxml:caption>
    <lxml:label>Business ph...</lxml:label>
    <lxml:description>Business Phone Number</lxml:description>
    <lxml:heading-1>Business Phone</lxml:heading-1>
    <lxml:heading-2>Number</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="STARTDTER">
  <lxml:caption>
    <lxml:label>Start date.....</lxml:label>
    <lxml:description>Start date (YYMMDD)</lxml:description>
    <lxml:heading-1>Start</lxml:heading-1>
    <lxml:heading-2>Date</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="TERMDATER">
  <lxml:caption>
    <lxml:label>Term. date.....</lxml:label>
    <lxml:description>Termination Date (YYMMDD)
</lxml:description>
    <lxml:heading-1>Term.</lxml:heading-1>
    <lxml:heading-2>Date</lxml:heading-2>
```

```

    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="DEPARTMENT">
  <lxml:caption>
    <lxml:label>Department.....</lxml:label>
    <lxml:description>Department Code</lxml:description>
    <lxml:heading-1> Dept</lxml:heading-1>
    <lxml:heading-2> Code</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="SECTION">
  <lxml:caption>
    <lxml:label>Section.....</lxml:label>
    <lxml:description>Section Code</lxml:description>
    <lxml:heading-1> Section</lxml:heading-1>
    <lxml:heading-2> Code</lxml:heading-2>
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="SALARY">
  <lxml:caption>
    <lxml:label>Salary.....</lxml:label>
    <lxml:description>Employee Salary</lxml:description>
    <lxml:heading-1>Salary</lxml:heading-1>
    <lxml:heading-2 />
    <lxml:heading-3 />
  </lxml:caption>
  <lxml:value />
</lxml:field>
<lxml:field name="MNTHSAL">
  <lxml:caption>
    <lxml:label>Monthly Salary</lxml:label>
    <lxml:description>Monthly Salary</lxml:description>
    <lxml:heading-1>Monthly</lxml:heading-1>

```

```

        <lxml:heading-2>Salary</lxml:heading-2>
        <lxml:heading-3 />
    </lxml:caption>
    <lxml:value />
</lxml:field>
<lxml:field name="STARTDTE">
    <lxml:caption>
        <lxml:label>Start date.....</lxml:label>
        <lxml:description>Start Date (DDMMYY)</lxml:description>
        <lxml:heading-1>Start</lxml:heading-1>
        <lxml:heading-2>Date</lxml:heading-2>
        <lxml:heading-3 />
    </lxml:caption>
    <lxml:value />
</lxml:field>
<lxml:field name="TERMDATE">
    <lxml:caption>
        <lxml:label>Term. date.....</lxml:label>
        <lxml:description>Termination Date (DDMMYY)
</lxml:description>
        <lxml:heading-1>Term.</lxml:heading-1>
        <lxml:heading-2>Date</lxml:heading-2>
        <lxml:heading-3 />
    </lxml:caption>
    <lxml:value />
</lxml:field>
</lxml:fields>
<lxml:lists>
    <lxml:list name="DEPTLIST" row-count="5">
        <lxml:list-header>
            <lxml:header name="DEPARTMENT">
                <lxml:heading-1> Dept</lxml:heading-1>
                <lxml:heading-2> Code</lxml:heading-2>
                <lxml:heading-3 />
            </lxml:header>
            <lxml:header name="DEPTDESC">
                <lxml:heading-1>Department</lxml:heading-1>
                <lxml:heading-2>Description</lxml:heading-2>
                <lxml:heading-3 />

```

```

    </lxml:header>
</lxml:list-header>
<lxml:list-entries>
  <lxml:entry>
    <lxml:column name="DEPARTMENT" id="DEPTLIST.0001.DEPTM
    <lxml:column name="DEPTDESC" id="DEPTLIST.0001.DEPTDI
  </lxml:entry>
  <lxml:entry>
    <lxml:column name="DEPARTMENT" id="DEPTLIST.0002.DEPTM
    <lxml:column name="DEPTDESC" id="DEPTLIST.0002.DEPTDI
  </lxml:entry>
  <lxml:entry>
    <lxml:column name="DEPARTMENT" id="DEPTLIST.0003.DEPTM
    <lxml:column name="DEPTDESC" id="DEPTLIST.0003.DEPTDI
  </lxml:entry>
  <lxml:entry>
    <lxml:column name="DEPARTMENT" id="DEPTLIST.0004.DEPTM
    <lxml:column name="DEPTDESC" id="DEPTLIST.0004.DEPTDI
  </lxml:entry>
  <lxml:entry>
    <lxml:column name="DEPARTMENT" id="DEPTLIST.0005.DEPTM
    <lxml:column name="DEPTDESC" id="DEPTLIST.0005.DEPTDI
  </lxml:entry>
</lxml:list-entries>
</lxml:list>
<lxml:list name="SECTLIST" row-count="5">
  <lxml:list-header>
    <lxml:header name="SECTION">
      <lxml:heading-1> Section</lxml:heading-1>
      <lxml:heading-2> Code</lxml:heading-2>
      <lxml:heading-3 />
    </lxml:header>
    <lxml:header name="SECDESC">
      <lxml:heading-1>Section</lxml:heading-1>
      <lxml:heading-2>Description</lxml:heading-2>
      <lxml:heading-3 />
    </lxml:header>
  </lxml:list-header>
<lxml:list-entries>

```

```

<lxml:entry>
  <lxml:column name="SECTION" id="SECTLIST.0001.SECTION"
  <lxml:column name="SECDESC" id="SECTLIST.0001.SECDESC
</lxml:entry>
<lxml:entry>
  <lxml:column name="SECTION" id="SECTLIST.0002.SECTION"
  <lxml:column name="SECDESC" id="SECTLIST.0002.SECDESC
</lxml:entry>
<lxml:entry>
  <lxml:column name="SECTION" id="SECTLIST.0003.SECTION"
  <lxml:column name="SECDESC" id="SECTLIST.0003.SECDESC
</lxml:entry>
<lxml:entry>
  <lxml:column name="SECTION" id="SECTLIST.0004.SECTION"
  <lxml:column name="SECDESC" id="SECTLIST.0004.SECDESC
</lxml:entry>
<lxml:entry>
  <lxml:column name="SECTION" id="SECTLIST.0005.SECTION"
  <lxml:column name="SECDESC" id="SECTLIST.0005.SECDESC
</lxml:entry>
</lxml:list-entries>
</lxml:list>
<lxml:json-list name="LIST01"><![CDATA[{"list":{
  "LIST01":{"header":[
    {"name":"DEPARTMENT","heading-2":" Dept","heading-3":"Code"},
    {"name":"DEPTDESC","heading-2":"Department","heading-
3":"Description"},
    {"name":"PCK105","heading-1":"Packed","heading-2":"
(10,","heading-3":"5)"),
    {"name":"DAT01","heading-2":"Date","heading-3":"field"},
    {"name":"BOOL1","heading-2":"Boolean","heading-3":"Field"},
    {"name":"FLT01","heading-2":"Float","heading-3":"field"},
    {"name":"FLT04","heading-2":"Float","heading-3":"4"},
    {"name":"INT01","heading-1":"Integer","heading-
2":"field","heading-3":"1"},
    {"name":"INT02","heading-1":"Integer","heading-
2":"field","heading-3":"2"}],
  "entries":[
    ["ADM","Admin > Dept",23456.78900,"2011-08-

```

```
29",true,+1.234567000000000E+004,+1.234567E+004,12345,12345],  
    ["SD","Sales\" & Dist",65432.12340,"2011-08-  
29",false,+1.234567000000000E+004,+1.234567E+004,12345,12345]  
    ]}}}  
  ]]>  
</lxml:json-list>  
</lxml:lists>
```

## Appendix C. Deprecated Weblets

When a weblet is enhanced, best efforts are made to ensure that it remains backwards compatible so that existing WAMs continue to look and behave as they always did. However, sometimes, changes in browser behavior, specific implementation requirements of new features or other technical restrictions make it impossible to implement new features or fixes and maintain backwards compatibility.

When this occurs, a new weblet is created and the old weblet is left unchanged. The new weblet will usually have the same display name but will have a different XSLT template name (often with something like "\_v2" appended to the name).

Weblets that have been deprecated are still shipped so that existing WAMs continue to work as before but they are normally removed from the Weblet Templates repository display. You can make them visible in the list by turning on the *Show deprecated Weblets* option in the XSL tab of the LANSAs Settings dialog.

If the changes to the weblet properties or behavior is significant then the documentation for the deprecated weblet is retained in this section.

Weblets in this category include:

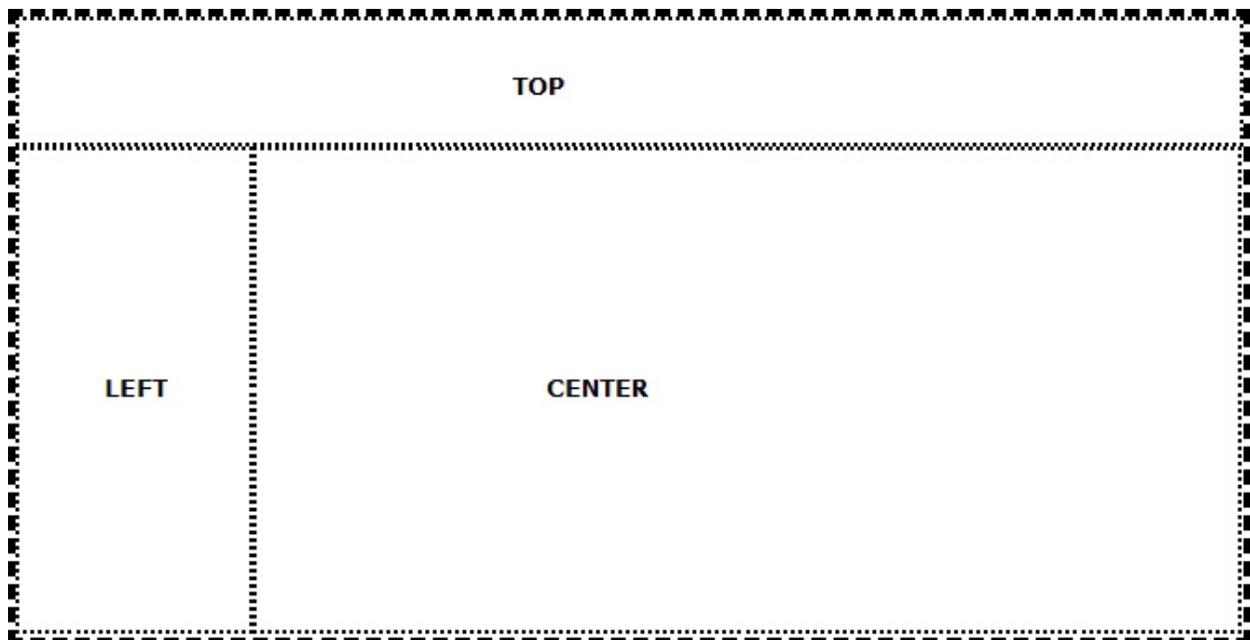
<b>Weblet name</b>	<b>Description</b>
<a href="#">Attachment Panel (std_attachment_panel)</a>	Panel with five areas where content can be dropped, Left, Top, Right, Center, and Bottom. Each of these has attachment layout manager behavior. Contents can be inserted or other weblets dropped into any of the five areas. Dropped weblets are sized according to attachment layout manager rules when they are dropped.
<a href="#">Banner (std_banner)</a>	Panel that scrolls content such as other weblets, text, and elements dropped into it.
<a href="#">Date (std_date)</a>	A text input box that supports the display, entry, prompting and validation of dates.
<a href="#">DateTime (std_datetime)</a>	A text input box that supports the display, entry, prompting and validation of date and/or time values.

<a href="#">Dynamic HTML menu bar (std_dhtml_menu)</a>	DHTML Multilevel Menu.
<a href="#">Push Button (std_button) &amp; Push Button with Images (std_image_button)</a>	A button with images. Images can be on the left or right, or both, of the caption.
<a href="#">Time (std_time)</a>	A text input box with added features to support the display, entry and validation of times.
<a href="#">Tree view (std_treeview)</a>	Tree control, Internet Explorer version only.
<a href="#">Tree view target (std_treeview_target)</a>	Panel that is a target of tree control selection actions.

## Attachment panel (std\_attachment\_panel)

The Attachment panel weblet provides a panel with five areas where content can be dropped: left, top, right, center, and bottom. Each of these has attachment layout manager behavior. Contents can be inserted or other weblets dropped into any of the five areas. Dropped weblets are sized according to attachment layout manager rules when they are dropped.

The following is an example of the appearance of a nearly empty attachment panel. In this example, just three of the attachment areas have been used. A thick dashed border has been specified for the attachment panel and thin dotted borders for the panels used in the three areas. The borders have been used for clarity in this example – you do not have to use visible borders and you may not wish to in your applications. Remember you can drag and drop other weblets (such as input boxes, check boxes and push buttons) onto each of the layout areas.



The attachment panel is one of a number of weblets that you can use to aid the creation of a consistent and visually appealing layout for your web pages. You may also wish to review the horizontal and vertical splitters and the panel and navigation panel weblets. This weblet (the attachment panel) is static – the user is not able to resize or otherwise manipulate the size and position of the panels that it contains.

## **QuickStart- Attachment panel**

To use the attachment panel you can follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Attachment panel weblet.
2. Drag and drop the weblet onto your page. Make sure the weblet is selected and then click on the Details tab. Set any properties required for the attachment panel, such as borders.
3. Now you can drag and drop or otherwise insert content into the required panes or layout areas. You may find it easiest to drag and drop the Panel weblet into each of the five layout areas that you wish to use. You can then more easily size those panels and insert other weblets onto those child panels.

## Properties - Attachment panel

The Attachment panel weblet's Properties are:

border	name
border_width	panes
class_top, class_left, class_center, class_right, class_bottom	pos_absolute
height	width
hide_if	

**name**

The name the weblet is identified with. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

An automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## **panes**

An XML node set specifying a set of panes to show. This is a system generated value set up when you drag the attachment panel onto the design view. You cannot modify the value of this property.

### **Default value**

document(")/\*/lxml:data/lxml:panes[@id='<unique id>'] (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

### **Valid values**

Not Applicable. (This value is system maintained.)

## **border**

The border style for the outer boundary of the weblet. For example 'dashed'.

### **Default value**

Blank (no border is shown).

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the pre-defined border styles. Note that 'window-inset' is only supported by Internet Explorer.

## **border\_width**

The width of the border for the outer boundary of the weblet. This property is ignored unless a border style is selected for the border property

### **Default value**

Blank. If a border style is selected, this default is equivalent to the 'medium' selection.

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

**'medium'**

**'thick'**

**'thin'**

**hide\_if**

An expression which, if evaluated to be True, will hide the weblet.

**Default value**

False() (that is, the weblet will always be shown)

**Valid values**

Any valid XPath expression that returns a Boolean value.

**class\_top, class\_left, class\_center,  
class\_right, class\_bottom**

These properties specify the Cascading Style Sheet (CSS) class names for the five layout areas of the weblet.

**Default value**

The name of the shipped class for the corresponding layout area of the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

### **Example**

In this example, Position Absolutely has been enabled for the weblet and the weblet was positioned as required in the Design view of the LANSAs Editor. This resulted in the value shown for the pos\_absolute\_design property.

 pos_absolute_design	'position:absolute;left: 324pt; top: 162.72pt;'
---	---

## **width**

The width of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

" (this specifies that the attachment panel will use the full width available in the containing element).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **height**

The height of the weblet on the web page.

Usually you would set the height and width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width-design and height\_design properties. However you can directly edit the property values if required.

### **Default value**

'250pt'

### **Valid values**

A height, in a valid unit of measurement, in single quotes.

## Banner (std\_banner)

### [QuickStart- Banner Properties - Banner](#)

The Banner weblet is essentially a marquee - a scrolling area of text - which can be added to any pane on a web page. A banner is implemented as a <marquee> HTML element.

Banners can be used to display information, or they can be an active element in the web page which will redirect you to another webpage when clicked. They are typically used for advertising or to display up to date information which changes regularly.

Firefox has a bug in the way it displays a banner that has been given a fixed size, has a "scroll" value of "scroll" or "slide" and is placed somewhere with a text alignment of "center" (such as a table cell or attachment panel). Firefox places the "box" for the banner in the correct place but calculates the location of the content assuming the banner is left aligned. The content is then only visible while the position of the content overlaps the position of the box.

## **QuickStart- Banner**

A banner is very simple to set up. When you open the XSL generated for the webroutine in the LANSAs Editor:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Banner weblet.
2. Drag the Banner weblet on to your web page where you want the banner to be displayed.
3. Click on the weblet to review the Details tab.
4. Set the value property to the appropriate field, system variable, multilingual variable or text literal which contains the text to display in the marquee.
5. If you want the text to be an active element in the page, set the on\_click\_wname or URL property to direct the click event to an appropriate page.

## Properties - Banner

The Banner weblet's Properties are:

class	pos_absolute	scroll_direction
disabled	presubmit_js	scroll_loop_count
formname	protocol	scroll_true_speed
height	reentryfield	show_in_new_window
hide_if	reentryvalue	target_window_name
name	scroll	URL
on_click_wamname	scroll_amount	value
on_click_wrname	scroll_delay	width
panes		

## **name**

The name of the banner. Normally, you would leave this as the default and let LANSa use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the banner.

## **Default value**

An automatically generated, unique identifier.

## **Valid values**

Single-quoted text.

## **Example**

This shows the default name:

name	concat('o', position(), '_LANSA_13186')
------	---

Or you can enter a unique name like:

name	'ABC_Banner'
------	--------------

**value**

The text string to be displayed on the banner.

**Default value**

Blank.

**Valid values**

Single-quoted text can be entered or the name of a multilingual text variable, system variable or field name (the ellipses button in the property sheet can be clicked to choose one from a list).

## **panes**

An XML nodeset specifying a set of panes to show. This is a system generated value set up when you drag the banner into a pane on the design view.

**Note:** This value cannot be modified and is for information only.

### **Default value**

document(")/\*//lxml:data/lxml:panes[@id='<unique id>'] (this is equivalent to the current pane where the unique id is an automatically generated identifier.)

### **Valid values**

Not Applicable. (This value is system maintained.)

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the banner will always be shown)

### Valid values

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

This example will hide the banner if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

hide_if	#STD_FLAG = 'X'
---------	-----------------

When the property loses focus, the expression is shown as follows:

hide_if	key('field-value', 'STD_FLAG') = 'X'
---------	--------------------------------------

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

Blank.

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

Blank.

**Valid values**

Any appropriate literal.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA' (that is, document.LANSA)

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## disabled

A Boolean property, the result of which determines whether the banner appears enabled or disabled.

The disabled attribute of the MARQUEE tag is only supported by Internet Explorer. The banner will ensure that mouse clicks are ignored for a disabled banner in all browsers, but the "grayed out" effect produced by the disabled attribute will only work in Internet Explorer.

### Default value

Blank – equivalent to False: the banner will always be enabled.

### Valid values

true(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

### Example

The following example will disable the banner if the field STD\_FLAG has a value of 'X'.

```
 disabled key('field-value', 'STD_FLAG') = 'X'
```

## **URL**

Indicates the URL to navigate to when the banner is clicked. Must be prefixed with a protocol (for example, `http://` or `https://`). This property should not be entered if a Webroutine has been nominated in the `on_click_wrname` property.

### **Default Value**

Blank

### **Valid Values**

A URL enclosed by single quotes.

**on\_click\_wamname**

The name of the WAM to be invoked when the banner is clicked. This property should not be entered if a URL has been nominated in the URL property.

**Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_click\_wrname**

The name of the Webroutine to be invoked when the banner is clicked. This property should not be entered if a URL has been nominated in the URL property but must be entered if the on\_click\_wamname is entered.

**Default value**

Blank.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, http:// or https://) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes, followed by a colon. This is usually 'http:' or 'https:'.

## **show\_in\_new\_window**

A Boolean property, the result of which determines whether response HTML from the banner click should be shown in a new browser window.

### **Default value**

false() – response HTML is shown in the current browser window.

### **Valid values**

True(), false() or any valid expression, involving field names, literals, XSL variables or JavaScript variables, which can be resolved to true() or false().

**target\_window\_name**

The name of the window, or frame, in which response HTML will be shown.

**Default value**

Blank – response HTML will be shown in the current window.

**Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be recognized. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**class**

The Cascading Style Sheet class to be applied to the banner.

**Default value**

'std\_banner'. This is the default class for the banner and is provided with the all shipped cascading styles sheets.

**Valid values**

Any valid class name selected from the current Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **scroll**

Controls how the text scrolls in the marquee.

### **Default value**

Blank – equivalent to Scroll.

### **Valid values**

Blank or 'alternate', 'scroll' or 'slide'

'Scroll' - The banner content scrolls in the direction specified by the *scroll\_direction* property. The text scrolls off the end of the banner and starts over.

'Slide' - The banner content scrolls in the direction specified by the *scroll\_direction* property. The text scrolls to the end of the banner and stops. Note that, in Firefox, the animation does not stop. The text will scroll off the end of the banner and start over, making 'slide' effectively the same as 'scroll'.

'Alternate' - The *scroll\_direction* reverses when the content reaches the edge of the banner.

## **scroll\_direction**

Controls the direction in which the text will flow.

### **Default Value**

Blank – equivalent to Left.

### **Valid Values**

Blank or 'down' (top to bottom), 'left' (left to right), 'right' (right to left) or 'up' (bottom to top).

'Down' - top to bottom

'Left' – right to left

'Right' – left to right

'Up' - bottom to top.

## **scroll\_loop\_count**

Controls the number of times a banner will play.

The "end state" of the banner after it has played *scroll\_loop\_count* times will depend on the value of the *scroll* property. For "scroll" the banner will be blank. For "slide" and "alternate" the text will remain visible at the position it finished.

The *scroll\_loop\_count* property is ignored by:

- Firefox
- all browsers if the *scroll* property is 'slide'.

### **Default Value**

Blank.

### **Valid Values**

Blank, -1 or any integer value.

Blank or -1 will cause the banner to loop indefinitely if the *scroll* property is blank, "scroll" or "alternate".

## **scroll\_amount**

Controls the number of pixels the text moves between each subsequent drawing of the weblet.

### **Default Value**

Blank. This allows the browser to set its own default. The default may vary slightly between browsers but should be around 6 pixels.

### **Valid Values**

Any integer value.

## **scroll\_delay**

Controls the speed of the scroll by specifying the delay, in milliseconds, between each update of the weblet.

To avoid overloading the client CPU, the browser will automatically round any value less than 60 milliseconds up to 60. The *scroll\_true\_speed* parameter can be used with Internet Explorer to override this behaviour and honor the specified scroll\_delay.

### **Default Value**

Blank. This allows the browser to set its own default. The default may vary slightly between browsers but should be around 85 milliseconds.

### **Valid Values**

Any integer value.

## scroll\_true\_speed

To avoid overloading the client CPU, the browser will automatically round any value of *scroll\_delay* less than 60 milliseconds up to 60. The *scroll\_true\_speed* parameter can be used to override this behaviour and honor the specified *scroll\_delay*.

*Scroll\_true\_speed* is currently only supported by Internet Explorer. Because of this, it is better practice to always use a value of 60 or greater for *scroll\_delay* and increase *scroll\_amount* to get the speed you require.

### Default Value

False().

### Valid Values

True(), false() or any valid expression, involving field names, literals or XSL variables, which can be resolved to true() or false().

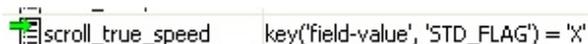
### Example

This example will use the appropriate scrolling properties as determined by the evaluation of the expression `#STD_FLAG is equal to 'X'`. The expression should be entered, and is shown when the property has focus, as follows:



The screenshot shows a property editor with a blue header bar containing the text "scroll\_true\_speed". To the right of the header bar is a text input field containing the expression "#STD\_FLAG = 'X'".

When the property loses focus, the expression is shown as follows:



The screenshot shows a property editor with a white header bar containing the text "scroll\_true\_speed". To the right of the header bar is a text input field containing the expression "key('field-value', 'STD\_FLAG') = 'X'".

## presubmit\_js

JavaScript code to be run prior to the submission of the form.

### Default value

Blank. No JavaScript is run.

### Valid values

Any valid JavaScript function, or JavaScript code followed by a semicolon (;).

If you want to execute the presubmit JavaScript only, without running the JavaScript that submits the request (thus canceling the onclick event), append **return false;** to your presubmit JavaScript.

### Example

The following example shows a message box:

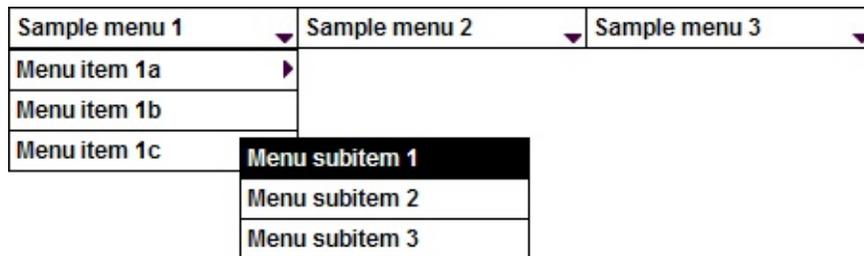
 presubmit_js	'alert("Hello world!");'
--	--------------------------

The following example shows a message box and cancels the submit JavaScript:

 presubmit_js	'alert("Hello world!"); return false;'
--	--

## Dynamic HTML menu bar (std\_dhtml\_menu)

The Dynamic HTML menu bar weblet provides the functionality of a menu bar that can invoke other web pages including other web routines. The menu bar can be arranged horizontally or vertically and the top level menu items can cause further menus to pop-up as the mouse moves over them. This is what the menu bar weblet looks like when arranged horizontally – in this example, two levels of popup menu are shown:



The weblet provides just four properties that affect its orientation and size. The menu items themselves are specified using the menu item designer.

## **QuickStart - Dynamic HTML menu bar**

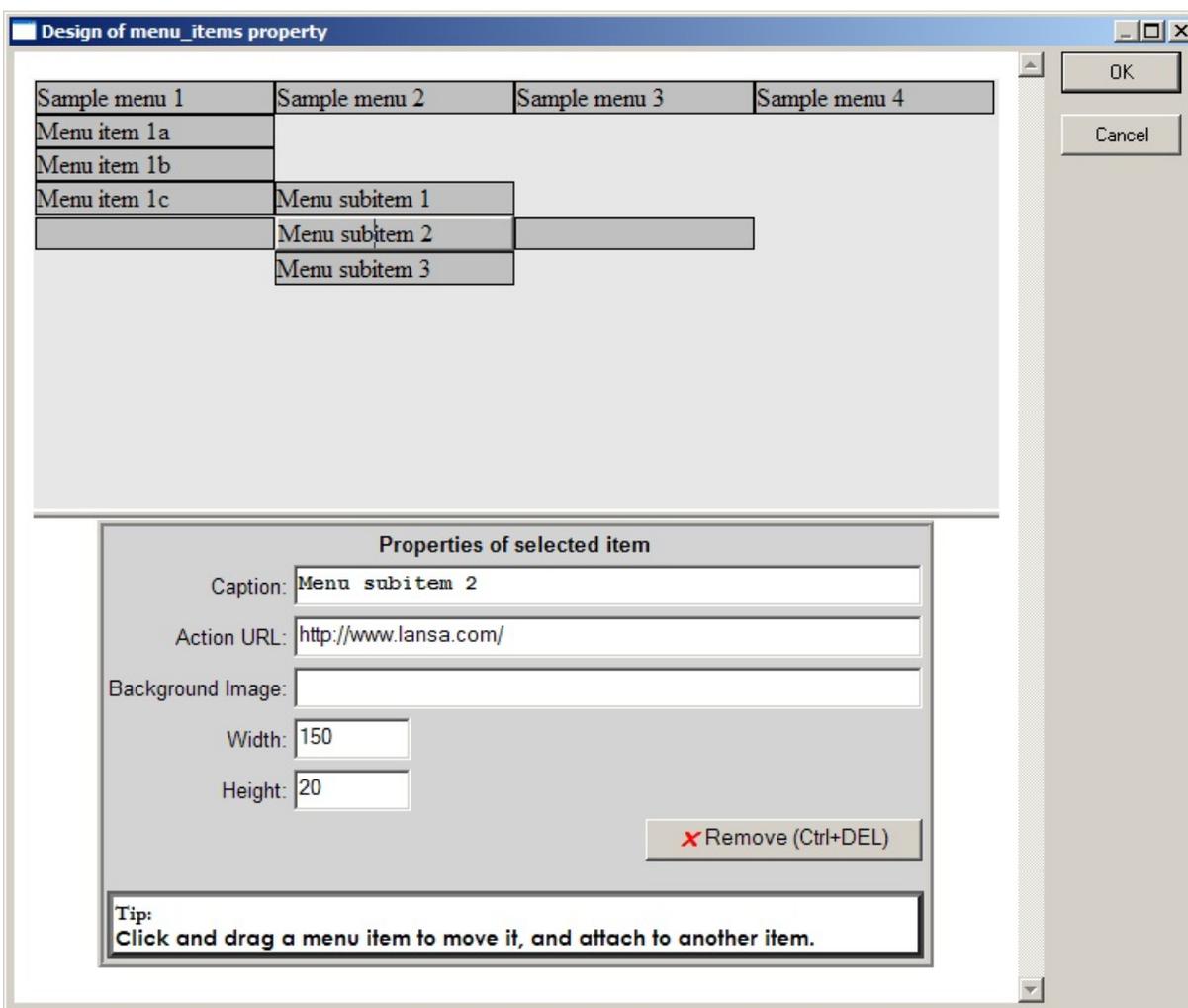
To use the Dynamic HTML menu bar weblet, open your webroutine in the LANSa Editor and follow these steps:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate the Dynamic HTML menu bar weblet.
2. Drag and drop the weblet onto your page in the Design view. Make sure the weblet on the page is selected and then click on the Details tab.
3. Click the ellipsis button next to the menu\_items property to open the menu items designer and define your menu items.

## Using the menu item designer

To open the menu item designer, follow these steps:

1. Make sure the Dynamic HTML menu bar weblet on your page is selected and then click on the Details tab.
2. Move the mouse pointer over the menu\_items property in the Details tab. An ellipsis should appear next to the property value. Click the ellipsis button to open the menu items designer. A window like the one shown below appears.



The top-half of the window shows a representation of the current state of the menu. Note that this is always shown in horizontal orientation irrespective of the current value of the orientation property. In this part of the window, you can:

- click on items to complete their details in the lower half;
- drag and drop items to rearrange them;

- add new items by clicking on the blank items shown adjacent to the currently selected item;
- remove items by selecting them and clicking the Remove button.

In the bottom-half of the window you can specify the details for the selected item as follows:

### **Caption**

Specifies the text that appears on the face of the menu item. You can also enter the text directly on the face of a menu item by clicking on it in the top-half of the window and typing.

### **Action URL**

Specifies a URL that the menu item will navigate to when clicked. You can specify a complete URL or one that is relative to the current page. The URL can invoke another webroutine by specifying an appropriate URL.

### **Background Image**

Specifies the path and file name of an image to be displayed as background to the menu item.

### **Width**

Specifies the width in pixels of the menu item. See Understanding menu bar and menu item width and height for information on how this value and the weblet width property determine the menu bar and item width.

### **Height**

Specifies the height in pixels of the menu item. See Understanding menu bar and menu item width and height for information on how this value and the weblet height property determine the menu bar and item height.

## **Understanding menu bar and menu item width and height**

The Dynamic HTML Menu bar weblet has width and height properties. In addition, you can specify the width and height of individual menu items in the menu item designer. Following is a summary of how these values work together to determine the width and height of the menu bar, the top-level items and the items in pop-up menus. The effect of these width and height values can also vary according to the chosen value for the orientation property of the weblet.

### **Width**

For menu items in pop-up menus, the width is determined by the width specified in the menu item designer for the first item in that pop-up. All items in the pop-up have the same width. The value specified for the width property of the weblet does not affect the width of these items.

For the top-level menu items (those that are shown statically on the page), the width is determined as follows:

If a value is specified for the weblet width property, then it applies to all top-level menu items (it overrides the width that may have been specified in the menu item designer for individual items).

Otherwise, the width depends on the orientation of the menu bar as follows:

- When in horizontal orientation ('top' is specified for the orientation property) the widths specified for individual top-level items is respected (that is, they can be different).
- When in vertical orientation ('left' or 'right' is specified for the orientation property) the width specified for the first top-level item applies to all the top-level items (that is, they are all the same width).

### **Height**

For menu items in pop-up menus, the height specified for individual items applies. That is the items can have different heights, both within one pop-up menu and across different pop-up menus. The value specified for the height property of the weblet does not affect the height of these items.

For the top-level menu items (those that are shown statically on the page), the menu item height depends on the orientation of the menu bar as follows:

- When in vertical orientation ('left' or 'right' is specified for the orientation property) the heights specified for individual top-level items is respected (that is, they can be different).

- When in horizontal orientation ('top' is specified for the orientation property) the height specified for the first top-level item applies to all the top-level items (that is, they are all the same height).

If a value is specified for the height property of the weblet, it does not alter the apparent height of the menu items (that is the dimensions of the visible boundary of the menu item). In other words, the top-level menu items appear to be the same size irrespective of the value of the height property. Instead, the height property specifies the vertical space reserved for the menu bar – that is, it affects the vertical spacing between the menu bar and following page elements.

By increasing the height property value you can increase the space between the menu bar and following page elements.

By decreasing the height property value you can decrease this space, even to the point that the menu bar can apparently overlap following page elements in some circumstances.

If no value is specified for the height property, the weblet allocates a default amount of space according to the height of the first or all top-level menu items, depending upon the orientation of the menu bar.

## **Properties - Dynamic HTML menu bar**

The Dynamic HTML menu bar weblet's properties are:

[menu\\_items](#)

[height](#)

[orientation](#)

[width](#)

## **menu\_items**

An XML nodeset that specifies the menu items. This is a system generated value set up when you drag the menu onto the design view.

Do not directly edit the value shown. Instead, click the ellipsis button to open the menu items designer. Refer to Using the menu item designer for more information.

### **Default value**

`document(")/*/lxml:data/lxml:menu[@id='<unique id>']`

where the <unique id> is an automatically generated identifier.

### **Valid values**

Not Applicable (this value is system generated and should not be modified).

## **orientation**

The orientation of the menu. This determines the positioning of the top-level menu items relative to each other and the direction or relative location that pop-up menus appear.

### **Default value**

'top'

### **Valid values**

Click the dropdown button next to this property in the property sheet to select one of the following values:

'top' The top-level menu items are arranged horizontally and first-level pop-up menus appear below the corresponding top-level menu item. Suitable for use as a horizontal menu bar across or near the top of the page.

'left' The top-level menu items are arranged vertically and first-level pop-up menus appear to the right of the corresponding top-level menu item. Suitable for use as a vertical menu bar on the left of the page.

'right' The top-level menu items are arranged vertically and first-level pop-up menus appear to the left of the corresponding top-level menu item. Suitable for use as a vertical menu bar on the right of the page.

## **height**

The height of the weblet on the web page. See Understanding menu bar and menu item width and height for information on how this property and the menu item height specified in the menu item designer determine the menu bar and item height.

### **Default value**

Blank (See Understanding menu bar and menu item width and height).

### **Valid values**

A height in pixels.

## **width**

The width of the top-level menu items on the web page. See Understanding menu bar and menu item width and height for information on how this property and the menu item width specified in the menu item designer determine the menu bar and item width.

### **Default value**

Blank (See Understanding menu bar and menu item width and height).

### **Valid values**

A width in pixels.

## **Push Button (std\_button) & Push Button with Images (std\_image\_button)**

The Push Button webllets provide windows-like push buttons for your web page. They look like this:



## **QuickStart- Push Button & Push Button with Images**

To add a push button to your web page:

1. Click on the Weblets tab, select Standard Weblets from the drop-down list near the top and locate either of the Push Button weblets.
2. Drag and drop the required weblet onto the web page. Click on the Details tab.
3. Set the caption to specify the text to be displayed on the button. In the case of the Push Button with Images, set the appropriate image properties. A left-hand side image and a right-hand side image can be set.
4. Set the `on_click_wrname` properties to the name of the webroutine to be invoked when the button is clicked. If the webroutine is in a different WAM to the current webroutine then you will also need to set the `on_click_wamname` property.

## Properties - Push Button & Push Button with Images

All these properties are common to both button weblets except for those indicated as *std\_image\_button only*.

caption	left_image_height	right_absolute_image_pa
class	left_relative_image_path	right_image_class
currentrowhfield	mouseover_class	right_image_height
currentrownumval	name	right_relative_image_pat
default_button	on_click_wamname	show_in_new_window
disabled	on_click_wrname	tab_index
formname	pos_absolute_design	target_window_name
height_design	presubmit_js	text_class
hide_if	protocol	title
left_absolute_image_path	reentryfield	width_design
left_image_class	reentryvalue	

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

`concat('o', position(), '_LANSA_n')` – this is the internal name given to the weblet by LANSA.

**Valid values**

A name enclosed in single quotes.

**caption**

The caption for the weblet.

**Default value**

'Caption'

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **currentrowhfield**

The field name to be used to post to the target webroutine the value that is specified in the currentrownumval property. The field name should be in single quotes.

See the description of the currentrownumval property for further information.

### **Default value**

'STDROWNUM'

### **Valid values**

Single-quoted text.

### **Example**

This example specifies the field name DEPTLINK as the field name to be used to post the value to the target webroutine. The target webroutine would need to have field DEPTLINK in its WEB\_MAP for \*BOTH or for \*INPUT in order to receive the value:

 currentrowhfield	'DEPTLINK'
--	------------

## **currentrownumval**

The value to post to the target webroutine in the field specified in the `currentrowhfield` property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

This property is used in conjunction with the `currentrowhfield` property to describe how to post values to a target webroutine. These two pieces of information are required to accomplish this:

1. `currentrowhfield`: the field name that the target webroutine uses to refer to the information
2. `currentrownumval`: a literal value or a field name in this (the source) webroutine that contains the necessary information

**Note:** Despite the name of the property being *currentrownumval*, the field name specified in `currentrownumval` is not required to be a numeric field.

### **Default value**

`position()`

### **Valid values**

Single-quoted text or the name of a field, system variable or multilingual text variable.

## **left\_relative\_image\_path**

*std\_image\_button only.*

The path and name, relative to the images directory, of the image to be displayed on the left of the weblet. If specified, the `left_absolute_image_path` property should be left blank.

### **Default value**

'ball\_red.gif'

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **left\_absolute\_image\_path**

*std\_image\_button only.*

The path and name of the image to be displayed on the left of the weblet. If specified, the `left_relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `left_relative_image_path` property.

### **Valid values**

The path and name of an image enclosed in single quotes.

## **left\_image\_height**

*std\_image\_button only.*

The height of the image on the left of the weblet.

### **Default value**

'12pt'

### **Valid values**

A height, in a valid unit of measurement, enclosed in single quotes.

## **right\_relative\_image\_path**

*std\_image\_button only.*

The path and name, relative to the images directory, of the image to be displayed on the right of the weblet. If specified, the `right_absolute_image_path` property should be left blank.

### **Default value**

Blank – by default, buttons do not display an image on the right.

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **right\_absolute\_image\_path**

*std\_image\_button only.*

The path and name of the image to be displayed on the right of the weblet. If specified, the `right_relative_image_path` property should be left blank.

### **Default value**

Blank – the default is to use the image specified in the `right_relative_image_path` property, if specified.

### **Valid values**

The path and name of an image, enclosed in single quotes.

## **right\_image\_height**

*std\_image\_button only.*

The height of the image on the right of the weblet.

### **Default value**

'12pt'

### **Valid values**

A height, in a valid unit of measurement, enclosed in single quotes.

## **reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

### **Default value**

'STDREENTRY'

### **Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'M'

**Valid values**

Any appropriate literal.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'X'. The expression should be entered, and is shown when the property has focus, as follows:

 hide_if	#STD_FLAG = 'X'
---	-----------------

When the property loses focus, the expression is shown as follows:

 hide_if	key('field-value', 'STD_FLAG') = 'X'
---	--------------------------------------

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**on\_click\_wamname**

The name of the WAM to be invoked when the weblet is clicked.

**Default value**

\$lweb\_WAMName (this is equivalent to the current WAM).

**Valid values**

The name of a WAM in single quotes. A list of known WAMs can be displayed by clicking the corresponding dropdown button on the property sheet.

**on\_click\_wrname**

The name of the Webroutine to be invoked when the weblet is clicked.

**Default value**

Blank – a Webroutine name must be specified.

**Valid values**

The name of a Webroutine in single quotes. The Webroutine must exist in the WAM specified in the on\_click\_wamname property. A list of known Webroutines can be displayed by clicking the corresponding dropdown button on the property sheet.

**protocol**

The protocol (for example, `http://` or `https://`) that should be used for navigation to the Webroutine specified in the `on_click_wname` property.

**Default value**

Blank. This is equivalent to the current protocol being used.

**Valid values**

A valid protocol, in single quotes. This is usually `'http:'` or `'https:'`.

## **show\_in\_new\_window**

A Boolean property, the result of which determines whether response HTML for the weblet should be shown in a new browser window.

### **Default value**

false() – response HTML is shown in the current browser window.

### **Valid values**

true(), false() or a valid expression.

## **target\_window\_name**

The name of the window, or frame, in which response HTML will be shown.

### **Default value**

Blank – response HTML will be shown in the current window.

### **Valid values**

The name of a window or frame, in single quotes. A list of known windows and frames can be displayed by clicking on the corresponding dropdown button in the property sheet, or a unique name can be entered.

'\_blank' will launch in a new window

'\_media' will launch a media panel in the current window

'\_search' will launch a search panel in the current window

'\_parent' will launch in the parent window (usually the current window)

'\_top' will launch in the top window (usually the current window)

Note that `_search` and `_media` are supported by Internet Explorer 6 only.

## **disabled**

A Boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

**text\_class**

*std\_image\_button only.*

The Cascading Style Sheet (CSS) class name of the text of the weblet.

**Default value**

The name of the shipped text class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**title**

Text to be displayed as a Tool Tip for the weblet when the mouse is hovered over it.

**Default value**

Blank – no Tool Tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_button' or 'std\_image\_button' - The name of the default shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **mouseover\_class**

The Cascading Style Sheet class name of the weblet when the mouse is moved over it.

### **Default value**

'std\_button\_mouseover' - The name of the default shipped mouseover class for the weblet.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **left\_image\_class**

*std\_image\_button only.*

The Cascading Style Sheet class name of the left image.

### **Default value**

Blank – the image is displayed without the application of a style.

### **Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

**right\_image\_class**

*std\_image\_button only.*

The Cascading Style Sheet class name of the right image.

**Default value**

Blank – the image is displayed without the application of a style.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## presubmit\_js

JavaScript code to be run prior to the submission of the form.

### Default value

Blank. No JavaScript is run.

### Valid values

Any valid JavaScript function, or JavaScript code followed by a semicolon (;).

If you want to execute the presubmit JavaScript only, without running the JavaScript that submits the request (thus canceling the onclick event), append **return false;** to your presubmit JavaScript.

### Example

The following example shows a message box:

 presubmit_js	'alert("Hello world!");'
--	--------------------------

The following example shows a message box and cancels the submit JavaScript:

 presubmit_js	'alert("Hello world!"); return false;'
--	--

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

## **default\_button**

A Boolean property, the result of which determines whether the button is the default button for the form. Only one button on the form can be the default button – setting to True will set all other buttons to False.

### **Default value**

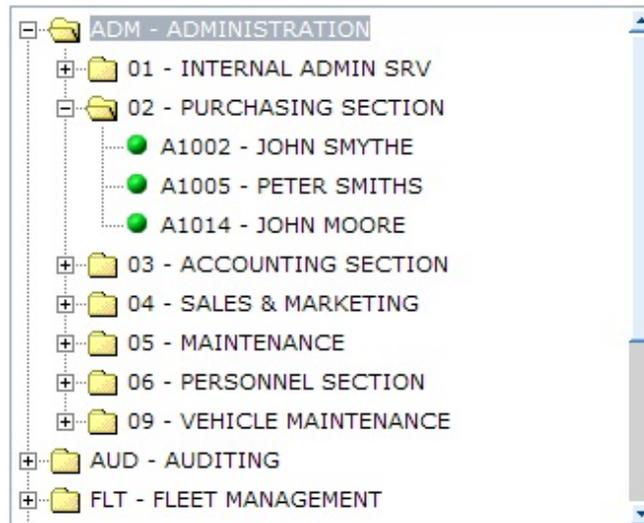
Blank - Equivalent to False.

### **Valid values**

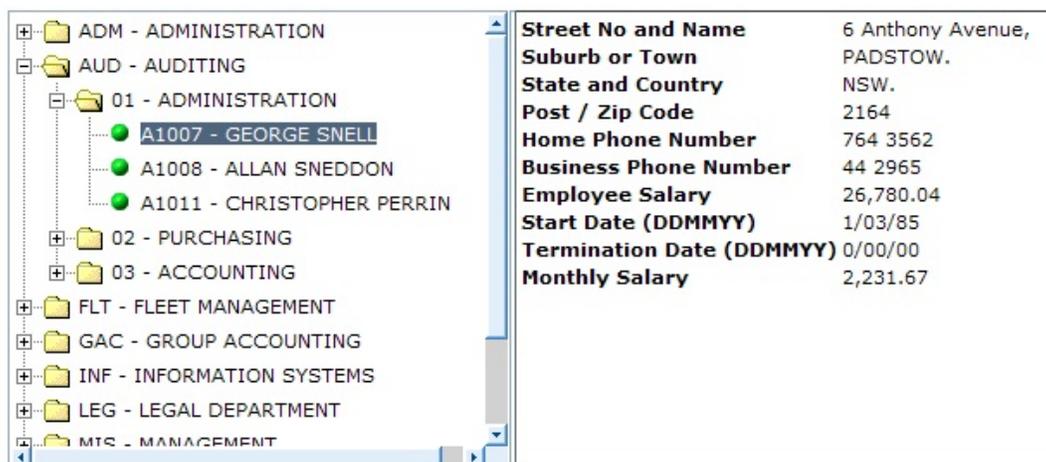
true(), false() or a valid expression.

## Tree View (std\_treeview)

The Tree View weblets provides a windows-like tree view on your web page. It can be used to produce something similar to this example:



It can also be used in conjunction with the Tree View Target (std\_treeview\_target) weblet, which responds to selection events from the tree view and can be used to display detail information. For example, the Tree View Target weblet can be seen displaying employee information to the right of the Tree View below:



## **QuickStart - Tree View**

Refer to [An In-Depth Look at the Tree View Weblet](#).

## Properties - Tree View

The Tree View weblet's properties are:

bg_color	list_is_selected_field	listname
default_style	list_onselect_wname_field	listname_of_parents_of_se
folder_closed_image	list_open_image_field	name
folder_open_image	list_parent_id_field	onexpand_wamname
formname	list_selected_style_field	pos_absolute_design
item_image	list_style_field	selected_style
list_caption_field	list_subitem_group_field	width_design
list_haschildren_field	list_tag_field	xmlid
list_image_field	list_type_field	xmltyped
list_is_expanded_field		

**name**

The name of the weblet. Normally, you would leave this as the default and let LANSAs use its own internal naming convention. When using this weblet in conjunction with the Tree View Target (this would normally be the case), it is recommended that a name be entered, as the Tree View Target will be required to reference it. Using this name will be clearer than using the LANSAs-generated name.

**Default value**

`concat('oTree', ancestor-or-self::lxml:list/@name,position())` – this is the internal name given to the tree view by LANSAs.

**Valid values**

A name in single quotes.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

**xmlid**

The identifier of the XML Data Island containing tree items. Leave blank to use the default id.

**Default value**

```
concat('xmltree_', $name)
```

**Valid values**

A name for the data island, in single quotes, or a valid JavaScript function that will produce a valid name (the default name uses the concat function).

## **xmltyped**

The identifier of the XML Data Island containing tree item types. Leave blank to use the default id.

### **Default value**

`concat('xmltreetype_', $name)`

### **Valid values**

A name for the data island, in single quotes, or a valid JavaScript function that will produce a valid name (the default name uses the concat function).

**folder\_closed\_image**

The path and file name, relative to the images directory, of an image that represents closed tree nodes.

**Default value**

'folder.gif'

**Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **folder\_open\_image**

The path and file name, relative to the images directory, of an image that represents open tree nodes.

### **Default value**

'folderopen.gif'

### **Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

**item\_image**

The path and file name, relative to the images directory, of an image to represent a leaf node of the tree.

**Default value**

'ball\_grn.gif'

**Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

**listname**

The name of the working list that contains the items used to populate the weblet.

**Default value**

Blank. A valid list name must be entered.

**Valid values**

The name of a valid working list, in single quotes. A list of valid list names can be chosen from by clicking the corresponding dropdown button in the property sheet.

**list\_caption\_field**

The name of the field in the listname working list that contains tree item captions.

**Default value**

Blank. A valid field name from the listname working list must be specified.

**Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

**list\_type\_field**

The name of the field in the listname working list that contains tree item types. Leave blank if not using types.

**Default value**

Blank. A valid field name from the listname working list should be specified if using types.

**Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_image\_field**

The name of the field in the listname working list that contains a tree item's image path and file name, relative to the images directory. Leave blank if using default images.

### **Default value**

Blank. Default images are used.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_open\_image\_field**

The name of the field in the listname working list that contains a tree item's image path and file name, relative to the images directory, that represents an expanded node. Leave blank if using default images.

### **Default value**

Blank. Default images are used.

### **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_tag\_field**

The name of the field in the listname working list that contains item tags. This is the non-visible, unique identifier of the tree item that can be used to identify it when selected or expanded.

## **Default value**

\$list\_caption\_field. The field used to store the tag information is the same field used for the caption.

## **Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

**list\_onselect\_wrname\_field**

The name of the field in the listname working list that contains the name of the Webroutine in the current WAM that is to be invoked when a tree item is selected.

**Default value**

Blank. A Webroutine will not be invoked when a tree item is selected.

**Valid values**

The name of a valid field, in single quotes. A list of fields in the listname working list can be chosen from by clicking the corresponding dropdown button in the property sheet.

## **list\_haschildren\_field**

The name of the field in the listname working list that determines whether a tree item has child items.

### **Default value**

'STD\_CODE'

### **Valid values**

The name of a valid field, in single quotes, that will contain a:

'Y' (the tree item has child items) or an

'N' (the tree item does not have child items).

## **list\_subitem\_group\_field**

The name of the field in the listname working list that will contain the depth (or level) of a selected item from the root. For example, a direct child of the root will have a depth of 2.

### **Default value**

'STD\_LEVEL'

### **Valid values**

The name of a valid numeric field, in single quotes.

## **list\_is\_selected\_field**

The name of the field in the listname working list, the value of which will determine if a tree item should be selected when displayed.

### **Default value**

Blank. Tree items cannot be pre-selected.

### **Valid values**

The name of the field in the working list that will contain a value of 'True' if an item in the tree should be selected. If set to 'Freeze', the item will be selected but the associated Tree View Target weblet (if applicable) will not be reloaded.

## **list\_is\_expanded\_field**

The name of the field in the working list that will control a tree item's expanded state.

### **Default value**

Blank. A tree item's expanded state cannot be controlled.

### **Valid values**

The name of the field in the working list that will contain a value of 'True' if an item should be expanded, and 'False' if it should not.

## **list\_parent\_id\_field**

The name of the field in the working list that will contain the identifier of the parent of the tree item.

### **Default value**

Blank. A tree item's parent can only be identified by using the listname\_of\_parents\_of\_selected property.

### **Valid values**

The name of the field in the working list that will contain the identifier of the parent of the tree item.

## **list\_style\_field**

The name of the field in the working list that controls a tree item's style. This allows the style of the child items to vary from the parent.

### **Default value**

Blank. The tree item adopts the default style.

### **Valid values**

The name of the field in the working list that will contain the Cascading Style Sheet style of the tree item.

**list\_selected\_style\_field**

The name of the field in the working list that controls a selected tree item's style.

**Default value**

Blank. The selected tree item adopts the default style.

**Valid values**

The name of the field in the working list that will contain the Cascading Style Sheet style of the tree item when it is selected.

**onexpand\_wamname**

The name of the WAM whose Webroutine will be invoked when an item in the tree is expanded.

**Default value**

Blanks. The current WAM will be invoked.

**Valid values**

The name of a WAM, in single quotes. A selection can be made from a list of known WAMs by clicking on the corresponding dropdown button in the property sheet.

## **onexpand\_wrname**

The name of the Webroutine that will be invoked when an item in the tree is expanded.

### **Default value**

Blank. The current Webroutine is the default.

### **Valid values**

The name of a valid Webroutine, in single quotes. A selection can be made from a list of valid Webroutines by clicking on the corresponding dropdown button in the property sheet.

## **listname\_of\_parents\_of\_selected**

The name of the working list returned to the WAM that will contain a list of parent identifiers for the currently selected or expanded tree item.

### **Default value**

Blank. A list of parent identifiers is not passed to the WAM.

### **Valid values**

The name of a valid working list, in single quotes. A selection can be made from a list of valid working lists by clicking on the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

Blank (weblet uses its default width).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_treeview' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **bg\_color**

The background color of the weblet.

### **Default value**

Blank. The background color will be the same as the web page.

### **Valid values**

A valid color, either in #RRGGBB format or a valid color name, in single quotes. A color can be chosen from a palette by clicking on the corresponding ellipses button in the property sheet.

### **Example**

Both of the following examples will set the background color to red:



A screenshot of a property sheet for the 'bg\_color' property. On the left, there is a small icon with three horizontal lines and a green square, followed by the text 'bg\_color'. To the right of this is a text input field containing the value '#FF0000'.

or



A screenshot of a property sheet for the 'bg\_color' property. On the left, there is a small icon with three horizontal lines and a green square, followed by the text 'bg\_color'. To the right of this is a text input field containing the value 'red'.

**default\_style**

The Cascading Style Sheet inline style of the tree control items.

**Default value**

Blank. The default style of the Cascading Style Sheet is used.

**Valid values**

A valid Cascading Style Sheet style name, in single quotes.

**selected\_style**

The Cascading Style Sheet inline style of the selected tree control items.

**Default value**

Blank. The default style of the Cascading Style Sheet is used.

**Valid values**

A valid Cascading Style Sheet style name, in single quotes.

## **An In-Depth Look at the Tree View Weblet**

The tree view weblet provides a windows-like tree view control for use by your WAM application. Following is a description of how it should be used. This description is rather long, and has been divided into these sections:

[About the Tree View Examples](#)

[What You Need to Know](#)

[When a Webroutine is Invoked](#)

[Accessing Key Information](#)

[The Tree View Target Weblet](#)

[Using a Navigation Panel](#)

[A Closer Look at WAMEX51](#)

[Invoking the WAM](#)

[Building the Tree View](#)

[The Role of the Tree View Target Weblet](#)

[Rebuilding the Tree View](#)

[Processing Expanding Tree Items](#)

[Accessing Ancestor Information](#)

As with all weblet controls that display a list of information, the Tree View Weblet is primarily driven by a working list in your WAM. Each entry in the working lists represents a tree item. Fields in the working list control the tree view item's appearance and behavior. These include the hidden key information of the tree item, its caption, what to do when the item receives focus and what to do when the tree item is expanded.

Depending on how you wish to utilize the tree view, different techniques are used. These are documented in What You Need to Know.

### **About the Tree View Examples**

The Tree View examples supplied are WAMEX50 and WAMEX51.

They are both simple examples of a tree that contains Department branches which, when expanded, show Section branches belonging to the Department, which, in turn, show Employees belonging to the Section.

Both examples use a Vertical Splitter weblet. This allows the Tree View weblet to be displayed on the left of the web page and details for a selected tree item to be displayed, in a Tree View Target weblet, on the right of the web page.

WAM01 is an enquiry-only example, whereas WAMEX51 allows the update of the selected department's description. Due to the updating of the tree, WAMEX51 uses additional Weblets and slightly different coding techniques to those of WAMEX50.

Both examples are intended to give you a good introduction to the Tree View weblet and how it interacts with other, related weblets, as well as the underlying WAM code.

## **What You Need to Know**

### **The Tree Working List**

In all cases, you will need a working list. Remember that each entry of the working list represents an item in the tree view. The working list should contain fields that will populate the list\_\* properties of the weblet. Refer to Properties - Tree View for a list of these. As a minimum, however, the working list should contain the following:

- A field to hold a caption for the tree item It can be any free-format text string that describes the tree item In the example WAMs, this is the Department description, the Section description, or the name of an Employee.
- A field to hold the key, or tag, of the tree item In the example WAMs, this field holds either the Department Code, Section Code, or Employee Number Composite keys are derived by querying another working list that holds ancestor items of a given item This is discussed in the Accessing Key Information section.
- A field to hold the name of the Webroutine to be invoked when a tree item is selected When a tree item is selected, you will usually want to perform some action, such as display the details associated with the selected item In the example WAMs, a Webroutine is invoked to display details of Departments, Sections and Employees Refer to the When a Webroutine is Invoked section for more information.
- A Sub-Item Group field This numeric field holds the nesting level of the tree item from its root item In the example WAMs, Departments have a level of 1, sections 2 and employees 3.
- A HasChildren field This is a single-byte field that should contain a Y or N. This denotes whether or not the tree item is a branch that can be expanded in order to display dependant information In the example WAMs, each Department has dependant Sections, so this field is set to Y Likewise, Sections have dependant Employees, so this field is also set to Y Employees,

however, do not have any dependant information, so this field is set to N.

### **The Parent Items Working List**

In most, if not all cases, your tree view will have multiple levels and, usually, will contain data that follows a parent/child pattern. In our examples, this is true of the Department/Sections/Employees data.

The Parent Items working list is passed into the WAM, providing a list of ancestors of an item that has been selected or is expanding. As such, it contains just one field, which must match the field used in the Tree View working list to hold a tree item's key information. More information about using this working list can be found later in this document.

### **When a Webroutine is Invoked**

#### **When a Tree Item is Selected**

The `list_onselect_wname_field` property of the tree view points to a field that contains the name of the Webroutine that is to be invoked when the corresponding item in the tree is selected. In the example WAMs, this is set to `ShowDepartmentDetail` for each Department entry.

When this Webroutine is invoked, certain information is passed into it:

The key information of the selected tree item. This is the field that is specified in the `list_tag_field` property. This can be used to access additional or related information that is relevant to the tree item. A `Web_map` to receive this field is required at all times.

The working list that contains the identifiers of the selected tree item's parents or ancestors. This is the list specified in the `listname_of_parents_of_selected` property. See the Accessing Key Information section for more information.

#### **When a Tree Item is Expanded**

The `onexpand_wname` property of the Tree View can be used to build the contents of a branch when it is expanded. In the example WAMs, when a Department or Section branch is expanded, the `TreeExpanding` Webroutine is invoked.

As well as the key information of the expanding tree item, the level of the tree item is passed into this Webroutine, along with the tree list itself and the list of parents of the expanding tree item. Based on the level of the expanding tree item, the appropriate method is invoked to add child items to the tree for the selected parent item.

So, a level of 1 indicates that a Department tree item is being expanded, and so

Sections belonging to that Department are added to the tree. A level of 2 indicates that a Section tree item is being expanded, and so Employees are added.

## **Accessing Key Information**

Because a tree item can have parent tree items, a working list to hold this information is required. It should be defined as containing just the field that is specified in the `list_tag_field` property of the tree.

It must be specified as a `Web_map` for the Webroutine that is used to display the main tree view working list. In WAMEX50, the ShowPage Webroutine has a `Web_map` specified for the list.

When passed into the WAM, it will have an entry for each of the selected item's parent items. Each entry contains the key information of the parent item. For tree items that do not have any parents, the list will be empty.

A `Web_map` to receive this list is only required if you wish to access information in it. This would typically be required for the selection Webroutine and the Tree Expanding Webroutine.

In the example WAMs, when a Section is selected, this list is used to ascertain the key value of the Department to which the Section belongs.

## **The Tree View Target Weblet**

Usually, you will want to display additional information relating to a tree item that has been selected.

In the example WAMS, when a Department is selected, the ShowDepartmentDetail Webroutine is invoked to show additional information for the Department.

There must be an area on the web page for this information to be shown. This is what the Tree View Target weblet is used for. When a tree item is selected, the Tree View Target associated with the tree (via the Tree View Target's `treeview_name` property) becomes active, or gets focus. The output from the Webroutine invoked when the tree item is selected is thus directed to the Tree View Target.

For more information on the Tree View Target (`std_treeview_target`)weblet, refer to its documentation.

## **Using a Navigation Panel**

In WAMEX50, which is a simple enquiry, the Tree View and Tree View Target weblets are displayed by the ShowPage Webroutine. This works fine for enquiry

purposes.

It becomes slightly more complex, however, if an update function for the details that are displayed in the Tree View Target weblet is introduced. For example, if the user changes the Department description and presses an update push button, you would expect the Tree View to be updated with the new description.

In order to get this to happen, the Tree View must be displayed in a different way to that shown in WAMEX50. Here, the Tree View is directly placed in the left-hand portion of the Vertical Splitter.

In WAMEX51, the left-hand portion of the Vertical Splitter contains a navigation panel. This panel is set to navigate to the DepartmentTree Webroutine. This is the Webroutine that contains the Tree View weblet. Separating it out like this means the Tree View can be easily refreshed on the web page, as it is, in effect, in its own sub-page of the main web page.

## A Closer Look at WAMEX51

To help reinforce the techniques described, this section contains sections of the WAM code, along with property settings of the Tree View and associated weblets.

### The List Definitions

The RDML code below shows the field and list definitions that are used by the Tree View weblet.

```
Define Field(#TreeID) Type(*CHAR) Length(256)
Define Field(#TreeCapt) Type(*CHAR) Length(256)
Define Field(#TreeLvl) RefId(#STD_LEVEL)
Define Field(#HasKids) Type(*CHAR) Length(1)
Define Field(#DetailWR) Type(*CHAR) Length(256)
Define Field(#Selected) Type(*CHAR) Length(10)

Def_List Name(#tvDepts) Fields(#TreeID #TreeCapt #TreeLvl #HasKids #DetailWR #Selected) Type(*Working)
  Entrys(9999)

Def_List Name(#Ancestors) Fields(#TreeID) Type(*Working)
```

The tvDepts working list is the list used to populate the Tree View. The function of its fields is as follows:

- TreeID, used to hold the key information for the tree item.
- TreeCapt, used to hold the caption for the tree item.
- TreeLvl, used to return the selected tree item's level to the WAM.
- HasKids, used to denote whether the tree item has children.
- DetailsWR, used to hold the name of the Webroutine to be invoked when the

tree item is selected.

- Selected, used to control whether a tree item should be selected.

The Ancestors list is used to receive parent tree item key information from the Tree View weblet for the selected tree item.

Right-click on the DepartmentTree Webroutine and select the LANSA Editor option. Once the LANSA Editor has opened, click on the tree itself and then select the Details tab. The properties of the tree view will be shown, as follows:

With Parameters	
name	'DeptTree'
formname	'LANSA'
xmlid	<i>concat('xmlTree_', \$name)</i>
xmltypeid	<i>concat('xmlTreeType_', \$name)</i>
folder_closed_image	'folder.gif'
folder_open_image	'folderopen.gif'
item_image	'person.jpg'
listname	'TVDEPTS'
list_caption_field	'TREECAPT'
list_type_field	
list_image_field	
list_open_image_field	
list_tag_field	'TREEID'
list_onselect_wname_field	'DETAILWR'
list_haschildren_field	'HASKIDS'
list_subitem_group_field	'TREELVL'
list_is_selected_field	'SELECTED'
list_is_expanded_field	
list_parent_id_field	
list_style_field	
list_selected_style_field	
onexpand_wamname	<i>/bxml:data/bxml:context/</i>
onexpand_wname	'TreeExpanding'
listname_of_parents_of_selected	'ANCESTORS'

Note the listname property contains the name of the tree view working list, tvDepts. Note also how the list\_\*, onexpand\_wname and listname\_of\_parents\_of\_selected properties relate back to fields in tvDepts.

## Invoking the WAM

The entry point of the WAM (that which should be used to execute the WAM via a browser URL or from another WAM) is the ViewDepartments Webroutine. It is designed to be the only entry point, to be executed only once. Any initialization logic for the WAM could be placed here. See below:

```

Webroutine Name(ViewDepartments) Onentry(*SESSIONSTATUS_NONE)

  * set the session status to active - this will ensure that session data is written out at the end of the routine.
  #com_owner.SessionStatus := Active

  * show the web page.
  Transfer Toroutine(ShowPage)

Endroutine

```

In this example, the only thing it has to do is set the Session Status to Active. This controls the writing out and reading in of any persistent session data when executing Webroutines in the WAM.

The ShowPage Webroutine is then executed. Again, this is designed to be the only place in the WAM used to display the web page. In this example, no Web\_maps specify what is to be displayed – all such information is handled by other Webroutines, as you will see. Non-field and list data has been specified via the LANSAs Editor, and so the ShowPage Webroutine shows as being 'empty'.

### Building the Tree View

1. Right-click on the ShowPage Webroutine and select the LANSAs Editor option.
2. Click on the left-hand side of the Vertical Splitter and select the Details tab to display the Navigation Panel's properties, as shown below:

With Parameters	
name	'TreeView'
border	
border_width	
hide_if	<i>false()</i>
pos_absolute	<i>'left:0pt;top:0pt;'</i>
width	<i>'100%'</i>
height	<i>'100%'</i>
size_panel_to_content	<i>false()</i>
size_panel_to_content_axis	<i>'both'</i>
scrolling	
class	<i>'std_nav_panel'</i>
transparent	<i>true()</i>
nav_url	
formname	
nav_wamname	<i>/bxml:data/bxml:context/ba</i>
nav_wrname	'DepartmentTree'
protocol	

Take note of the nav\_wrname property. This is the Webroutine that is invoked whenever the Navigation Panel is displayed. In this example, it

points to the DepartmentTree Webroutine, which is used to build the tree view.

3. Close the XML editor and look at the DepartmentTree Webroutine in the WAM source, as shown below:

```
Webroutine Name(DepartmentTree)
  Web_Map For(*both) Fields((#tvDepts *PRIVATE))
  Web_Map For(*both) Fields((#Ancestors *PRIVATE))

  * build the tree if requested (this will happen the first time through).
  If (#BuildTree)
    #com_owner.BuildDepartmentList

    #BuildTree := False
  Endif
Endroutine
```

Note the Web\_map definitions. Because this Webroutine is used to display the tree view, it has Web\_maps for the working list that represents the tree, along with the working list that is used to hold ancestor items for a selected tree item. Wherever the tree view working list is specified as a Web\_map, the ancestor list must also be specified as a Web\_map. If it isn't, your WAM won't work correctly.

BuildTree is a Boolean field that is used to control the building of the tree view. Its default (in its field definition, at the top of the source) is True, which means that when this Webroutine is invoked for the first time, by the Navigation Panel, the tree view will be built. BuildTree is then set to False, ensuring that the tree is not rebuilt on subsequent invocations.

### **BuildDepartmentList method:**

```
Mthroutine Name(BuildDepartmentList)

  Clr_List Named(#tvDepts)

  Select Fields(#DEPARTMENT #DEPTDESC) From_File(DEPTAB)

    #com_owner.AddListEntry I_Caption(#DEPARTMENT.BlankConcat(' ', #DEPTDESC)) I_Identifier(#DEPARTMENT)
    I_Level(1) I_Haschildren(Y) I_Detailwebroutine(ShowDepartmentDetail)
  Endselect
Endroutine
```

Note the setting for each entry to be added to the Tree View working list. Refer to List Definitions in [A Closer Look at WAMEX51](#) for a full explanation of

these.

## The Role of the Tree View Target Weblet

1. Right-click on the ShowPage Webroutine again and select the LANSA Editor option.
2. Click on the right-hand side of the Vertical Splitter and then click the Details tab. The properties for the Tree View Target will be displayed.

Note the `treeview_name` property points to the name of the Tree View weblet as defined in the DepartmentTree Webroutine. This indicates that the Tree View Target will receive selection events from the tree view. Effectively, it will become the active, target portion of the web page when something is selected in the tree.

This, in combination with the `list_onselect_wrname_field` property of the tree view, will display details of a selected Department, Section or Employee.

3. Close the LANSA Editor.

## DepartmentDetail Webroutine in the WAM source:

```
Webroutine Name(ShowDepartmentDetail)
  Web_Map For(*input) Fields(#TreeID)
  Web_Map For(*output) Fields((#DEPARTMENT *HIDDEN) #DEPTDESC)

  #DEPARTMENT := #TreeID

  #SelID := #DEPARTMENT

  Fetch Fields(#DEPTDESC) From_File(DEPTAB) With_Key(#DEPARTMENT)

Endroutine
```

This is the Webroutine that is invoked when a Department is selected in the tree view. Note the `*input Web_map`. This is the field specified against the `list_tag_field` property of the list view weblet. It contains the identifier, or key, of the selected tree item.

The `SelID` field is defined as a persistent session field (`Web_map` with `*NONE` and `*PERSIST`) and is used to hold the selected key on multiple invocations of the WAM. This is used when rebuilding the tree.

When the Webroutine ends, because the active portion of the web page is the Tree View Target, the output from the Webroutine is directed to it, so you see the department details in the right-hand portion of the Vertical Splitter.

## Rebuilding the Tree View

1. Right-click on the ShowDepartmentDetail Webroutine and select the LANSAs Editor option. Note that, as well as the department description, an Update push button is displayed. Click on it and select the Details tab. Its properties will be displayed, as follows:

With Parameters	
name	concat('o', position(), '_LANSA_5927')
caption	'Update'
reentryfield	'STDREENTRY'
reentryvalue	'M'
hide_if	false()
formname	'LANSA'
pos_absolute_design	'left:0pt;top:0pt;'
width_design	'0pt'
height_design	'0pt'
on_click_wamname	/bxml:data/bxml:context/bxml:webaj
on_click_wrname	'UpdateDepartment'
protocol	
show_in_new_window	false()
target_window_name	'TreeView'

Note the on\_click\_wrname and target\_window\_name properties. The on\_click\_wrname property contains the name of the Webroutine to invoke when the push button is clicked.

The target\_window\_name property is used to direct the output of the WAM to a specified window. In effect, this becomes the active portion of the web page. In this example, the Navigation Panel will be the target window.

2. Close the LANSAs Editor.

### UpdateDepartment Webroutine in the WAM source:

```

Webroutine Name(UpdateDepartment)
  Web_Map For(*input) Fields(#DEPARTMENT #DEPTDESC)

  Update Fields(#DEPTDESC) In_File(DEPTAB) With_Key(#DEPARTMENT)

  * force the tree to be rebuilt.
  #Buildtree := True

  * rebuild and re-display the tree.
  Transfer Toroutine(DepartmentTree)
Endroutine

```

Note that the BuildTree boolean field is set to True, indicating that the tree should be rebuilt. Control is then transferred to the DepartmentTree Webroutine, from where the tree view is rebuilt. Have another look at the AddListEntry method:

```

Mthroutine Name(AddListEntry)
  Define_Map For(*input) Class(#PRIM_ALPH) Name(#_Caption)
  Define_Map For(*input) Class(#PRIM_ALPH) Name(#_Identifier)
  Define_Map For(*input) Class(#PRIM_NMBR) Name(#_Level)
  Define_Map For(*input) Class(#PRIM_ALPH) Name(#_HasChildren)
  Define_Map For(*input) Class(#PRIM_ALPH) Name(#_DetailWebroutine)

  #TreeCapt := #_Caption
  #TreeID := #_Identifier
  #TreeLvl := #_Level
  #HasKids := #_HasChildren
  #DetailWR := #_DetailWebroutine

  If (#TreeID *EQ #SelID)
    #SELECTED := Freeze
    #SelID := *BLANKS
  Else
    #SELECTED := *BLANKS
  Endif

  Add_Entry To_List(#tvDepts)
Endroutine

```

Note the If/Else/Endif code. Remember the SelID field? If the entry being added to the tree is the same one that has just been updated, the SELECTED field is set to 'freeze'. SELECTED is a field in the tree view working list which drives the list\_is\_selected\_field property of the tree view. Setting it to freeze does two things: it pre-selects the tree view item and it stops the TreeViewTarget from being reloaded.

## Processing Expanding Tree Items

Open the DepartmentTree Webroutine in the LANSAs Editor and select the tree view weblet. Click the Details tab to display its properties. Note the onexpand\_wrname property is set to TreeExpanding. This is the Webroutine to be invoked when an expander (+) of a tree item is clicked on. Close the LANSAs Editor and have a look at the TreeExpanding Webroutine in the WAM source:

```

Webroutine Name(TreeExpanding)
  Web_Map For(*input) Fields(#TreeID)
  Web_Map For(*input) Fields(#TreeLvl)
  Web_Map For(*both) Fields((#tvDepts *PRIVATE))
  Web_Map For(*input) Fields((#Ancestors *PRIVATE))

  Case (#TreeLvl)
    * a department is expanding - add sections to list.
  When (*EQ 1)
    #com_owner.BuildSectionsList( #TreeID )

    * a section is expanding - add employees to list.
  When (*EQ 2)
    #SECTION := #TreeID
    #DEPARTMENT := #com_owner.GetAncestor( 1 )

    #com_owner.BuildEmployeesList( #DEPARTMENT, #SECTION )
  Endcase

  * re-display the tree.
  Transfer Toroutine(DepartmentTree)
Endroutine

```

Note the \*input field Web\_maps: the TreeID field, which holds the key information of the expanding tree item, and the TreeLvl field, which indicates the level of the expanding tree item. The tree view working list is also passed in, along with the list of ancestors of the expanding tree item. Note that the tree view working list is specified as \*both – it will be passed to the DepartmentTree Webroutine at the end of the TreeExpanding routine. Also, the ANCESTORS list is \*input – it is only needed by this routine.

The Case statement determines what should be added to the tree view working list. If the level is 1, it means a Department tree item is being expanded, so Sections need to be added. If it's 2, a Section is being expanded and Employees need to be added. Once entries have been added to the tree working list, control is transferred back to the DepartmentTree Webroutine which displays the tree.

In [Accessing Ancestor Information](#), the focus is on what happens if a Section tree item is being expanded in order to show what you need to do to retrieve parent key information.

## Accessing Ancestor Information

Refer back to [Processing Expanding Tree Items](#). When processing level 2, the SECTION field is set to the incoming identifier (this was set when the Sections

were added to the tree view working list in the BuildSectionsList method). Of course, a Section has a parent of Department.

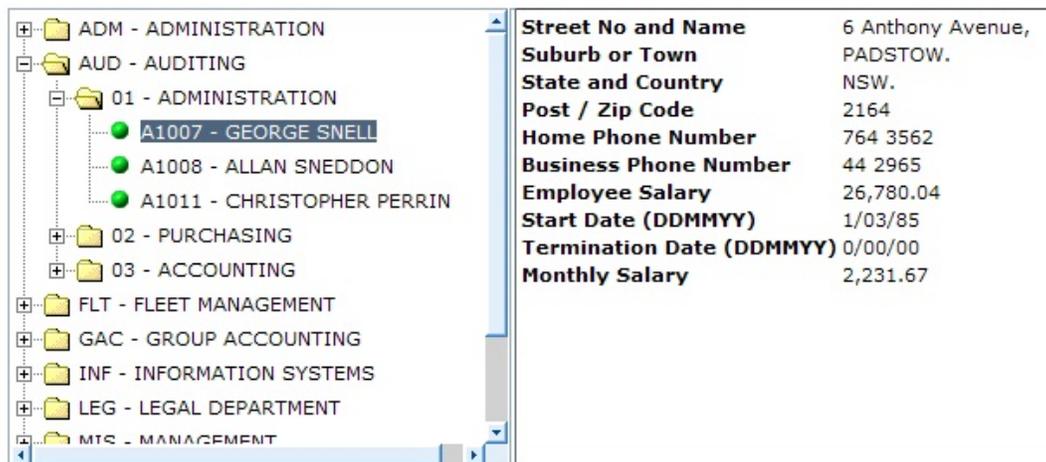
In order to determine the Department to which the Section being expanded belongs, the ANCESTORS working list is used. This list contains multiple entries of the TreeID field. A Section only has **one** parent, so the ancestor working list will contain **one** entry. In this example, the GetAncestor method is executed to retrieve the key value of the Section's parent.

If the tree contained more levels, the ANCESTOR working list would contain more than one entry for a level three or higher item, and so multiple Get\_entries could be used to build up the full list of parent keys.

## Tree View Target (std\_treeview\_target)

The Tree View Target weblet is a container-type control used in conjunction with the Tree View weblet. It responds to selection events from the Tree View and, as such, can be used to display information based on that selection.

The right-hand portion of the image below shows it in action (showing information for the selected employee in the tree view):



The screenshot displays a weblet interface. On the left, a tree view shows a hierarchy of folders: ADM - ADMINISTRATION, AUD - AUDITING, 01 - ADMINISTRATION, 02 - PURCHASING, 03 - ACCOUNTING, FLT - FLEET MANAGEMENT, GAC - GROUP ACCOUNTING, INF - INFORMATION SYSTEMS, LEG - LEGAL DEPARTMENT, and MTS - MANAGEMENT. Under '01 - ADMINISTRATION', three employees are listed: A1007 - GEORGE SNELL (selected), A1008 - ALLAN SNEDDON, and A1011 - CHRISTOPHER PERRIN. On the right, a table displays the details for the selected employee, A1007 - GEORGE SNELL.

<b>Street No and Name</b>	6 Anthony Avenue,
<b>Suburb or Town</b>	PADSTOW.
<b>State and Country</b>	NSW.
<b>Post / Zip Code</b>	2164
<b>Home Phone Number</b>	764 3562
<b>Business Phone Number</b>	44 2965
<b>Employee Salary</b>	26,780.04
<b>Start Date (DDMMYY)</b>	1/03/85
<b>Termination Date (DDMMYY)</b>	0/00/00
<b>Monthly Salary</b>	2,231.67

## **QuickStart - Tree View Target**

An example of how the Tree View Target weblet interacts with the Tree View weblet is included in [An In-Depth Look at the Tree View Weblet](#).

## Properties - Tree View Target

The Tree View Target weblet's properties are:

<code>treeview_name</code>	<code>wamname</code>	<code>tag_fieldname_alias</code>
<code>formname</code>	<code>wrname</code>	<code>resize_to_content</code>
<code>pos_absolute_design</code>	<code>reentryfield</code>	<code>class</code>
<code>width_design</code>	<code>reentryvaluereentryfield</code>	<code>bg_colortreeview_name</code>
<code>height_designbg_color</code>		

## **treeview\_name**

The name of the tree view weblet this tree view target is to be associated with. Selection events fired by the tree view will be heard by this target. The result of this is that the Tree View Target becomes the 'active' panel, and will have content directed to it by the next Webroutine that is run.

### **Default value**

`concat('oTree', ancestor-or-self::lxml:list/@name,position())` – this is the internal name given to the tree view by LANSA.

### **Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

**formname**

The name of the HTML form that is posted to the server.

**Default value**

'LANSA'

**Valid values**

A name for the form, in single quotes. A list of known form names is available by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute\_design**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (not positioned).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

**width\_design**

The width of the weblet on the web page.

**Default value**

'100%' (this is equivalent to the weblet adopting the width of its container).

**Valid values**

A width, in a valid unit of measurement, in single quotes.

**height\_design**

The height of the weblet on the web page.

**Default value**

Blank (weblet uses its default height).

**Valid values**

A height, in a valid unit of measurement, in single quotes.

**wamname**

The name of the WAM to be invoked when an item in the associated tree view is selected.

**Default value**

Blank – the current WAM is invoked, but only if the wrname property is specified.

**Valid values**

A valid WAM name, in single quotes. To choose from a list of known WAMs, click the corresponding dropdown button in the property sheet.

**wrname**

The name of the Webroutine to be invoked when an item in the associated tree view is selected.

**Default value**

Blank – the current WAM is invoked, but only if the wrname property is specified.

**Valid values**

A valid WAM name, in single quotes. To choose from a list of known WAMs, click the corresponding dropdown button in the property sheet.

**reentryfield**

The field name to be used to post to the WAM the value that is specified in the reentryvalue property. The field name should be in single quotes.

**Default value**

'STDREENTRY'

**Valid values**

Any repository- or WAM-defined field name. A list of known field names is available by clicking the corresponding dropdown button in the property sheet.

**reentryvalue**

The value to post into the field specified in the reentryfield property. If that field is alphanumeric, the value must be specified in single quotes. If it is numeric, the value can be specified with or without quotes.

**Default value**

'D'

**Valid values**

Any appropriate literal.

## **tag\_fieldname\_alias**

When a Webroutine is invoked to navigate to a page for this panel, the field posted to it is the tag field, as specified in the list\_tag\_field property of the associated tree view. If the field required by the target Webroutine is different to the tag field name, it can be specified here.

### **Default value**

Blank – the field name specified in the list\_tag\_field property of the associated tree view is used.

### **Valid values**

A valid field name, in single quotes. Click the corresponding dropdown button in the property sheet to choose from a list of known field names.

## **resize\_to\_content**

A Boolean property that indicates whether the panel will be resized to the content size of the page navigated to.

### **Default value**

true() – the panel will be resized.

### **Valid values**

true(), false() or a valid expression.

**class**

The Cascading Style Sheet class name of the weblet.

**Default value**

'std\_treeview\_target' - The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **bg\_color**

The background color of the weblet.

### **Default value**

Blank. The background color will be the same as the web page.

### **Valid values**

A valid color, either in #RRGGBB format or a valid color name , in single quotes. A color can be chosen from a palette by clicking on the corresponding ellipses button in the property sheet.

### **Example**

Both of the following examples will set the background color to red:

 bg_color	'#FF0000'
--	-----------

or

 bg_color	'red'
--	-------

## Date (std\_date)

### QuickStart - Date Properties - Date

The date weblet provides a text input box control with added features to support the display, entry, prompting and validation of dates. It broadly corresponds to the `<input type="text">` HTML element.

An example of the weblet is shown below. In this example, the (optional) calendar prompt button has been clicked and the calendar prompt window is visible.



The date weblet is best used with fields of type date. If you use this type, the data will automatically be passed in the format expected by the weblet. You can use the date weblet with fields of other numeric types such as packed or signed, but it is your responsibility to ensure the numeric value is formatted in the correct ISO format expected by the date weblet. For example, you could use a signed (8, 0) field containing a date in YYYYMMDD format with an edit word ('0 - - ') to format it as an ISO date format.

## QuickStart - Date

For date fields that have `std_date` as their default visualization, you do not need to manually add them to your web page. Simply include your date fields in your `web_map` or in a list that is present in your `web_map` and they will be visualized using the date weblet. Similarly fields of type time and of type datetime will be visualized using the time (`std_time`) and datetime (`std_datetime`) weblets.

If you do need to add the date weblet to your page manually, simply drag the date field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Date weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines `web_map`.
4. You may also wish to set the `date_mask` property as required.

## Properties - Date

The Date weblet's properties are:

allow_sqlnull	display_mode	read_only
button_image	hide_calendar	tab_index
button_title	hide_if	title
class	name	value
date_mask	onchange_script	width
disabled	pos_absolute	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The property name is 'value' and its value is '#V1DAT04'. The editor has a dark blue header bar with the property name and a white input field with the value.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The property name is 'value' and its value is 'key('field-value', 'V1DAT04')'. The editor has a light gray header bar with the property name and a white input field with the value.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

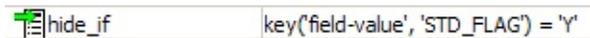
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value '#STD\_FLAG = 'Y'' is entered in a text field.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is in a text field. To its right, the value 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in another text field.

## **allow\_sqlnull**

A Boolean property which determines if the date value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## **date\_mask**

Specifies the format or mask used to display and enter the date for the weblet. This is a string containing a number of format specifiers that tell the weblet how to format the date.

See the DateTime weblet for a full list of valid format specifiers.

**Note:** this specifies the presentation format the weblet uses. The input and output date received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

### **Default value**

'YYYY-MM-DD'

### **Valid values**

Any string containing valid format specifiers.

**button\_image**

The path and file name, relative to the images virtual directory, of the image to display on the calendar prompt button.

**Default value**

'fp\_im003.gif' (this image is shipped with LANSA).

**Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The `tab_index` property value determines the tab order as follows:

1. Objects with a positive `tab_index` are selected in increasing `tab_index` order (and in source order to resolve duplicates).
2. Objects with a `tab_index` of zero or blank (the default) are selected in source order.
3. Objects with a negative `tab_index` are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

### **Example**

 tab_index	3
---	---

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**button\_title**

Specifies text for the calendar button (if shown) that may display as tip text as the mouse moves over the button.

**Default value**

Blank – the text specified for the title property is used.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

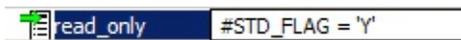
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''. The dropdown menu has a small icon on its left side.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression 'key('field-value', 'STD\_FLAG') = 'Y''. The dropdown menu has a small icon on its left side.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **hide\_calendar**

A boolean property, the result of which determines whether the calendar button is shown for the weblet.

### **Default value**

Blank – equivalent to false (that is, the calendar button will be shown).

### **Valid values**

true(), false() or a valid expression.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page. The weblet will reserve a minimum width based on the data to be displayed.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

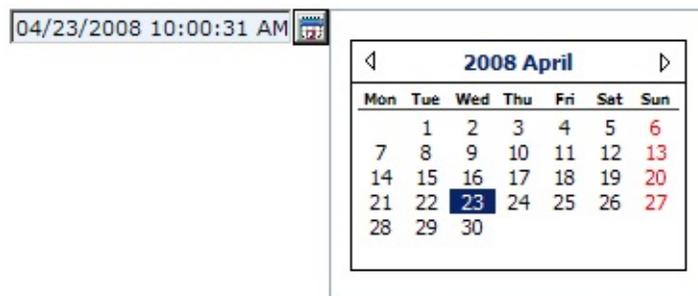
Any valid JavaScript statement(s).

## DateTime (std\_datetime)

### [QuickStart - DateTime Properties - DateTime](#)

The datetime weblet provides a text input box control with added features to support the display, entry, prompting and validation of date and/or time values. It broadly corresponds to the `<input type="text">` HTML element.

An example of the weblet is shown below. In this example, the (optional) calendar prompt button has been clicked and the calendar prompt window is visible:



The datetime weblet is used to display and receive input for fields containing dates, times or datetimes. If your field contains only a date or only a time you may prefer to use one of the specialized weblets that are based on this weblet: `std_date` or `std_time`.

The datetime weblet is best used with fields of date, time or datetime data types. If you use these types, the data will automatically be passed in the format expected by the weblet. You can use the datetime weblet with fields of other numeric types such as packed or signed, but it is your responsibility to ensure the numeric value is formatted in the correct ISO format expected by the datetime weblet. For example, you could use a signed (14, 0) field containing a date and time in YYYYMMDDHHMMSS format with an edit word ('0 - - & : : ') to format it as an ISO date and time format.

## QuickStart - DateTime

For datetime fields that have datetime weblet as their the default visualization, you usually do not need to manually add it to your web page. Simply include your datetime fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the datetime weblet. Similarly fields of type date and of type time will be visualized using the date (std\_date) and time (std\_time) weblets.

If you do need to add the datetime weblet to your page manually, simply drag the datetime field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSA Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Datetime weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.
4. You may also wish to set the size and date\_mask properties as required.

## Properties - DateTime

The DateTime weblet's properties are:

allow_sqlnull	display_in_utc	read_only
button_image	hide_calendar	size
button_title	hide_if	tab_index
class	input_type	time_mask
date_mask	name	title
disabled	onchange_script	value
display_mode	pos_absolute	width

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in a dark blue box, and its value is '#V1DAT04'.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in a light blue box, and its value is 'key('field-value', 'V1DAT04')'.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

**size**

The size of the weblet data in characters. – the browser sizes the input box according to the number of characters specified, but will reserve a minimum width based on the data to be displayed. Sizing the weblet by dragging the grab handles (or manually specifying the width property) can increase the width but not reduce it beyond the minimum that the weblet determines.

**Default value**

24.

**Valid values**

A numeric value. Usually you set this according to the field definition. If the weblet is to display only a date, specify 10. If the weblet is to display only a time, specify 11 or greater. If the weblet is to display a date and time, specify 22 or greater. When a time is included in the data, remember to allow room for the AM/PM indicator.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

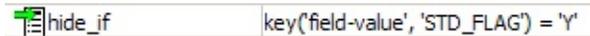
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is '#STD\_FLAG = 'Y''. To the left of the property name is a small icon representing a document with a green checkmark.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. The value entered is 'key('field-value', 'STD\_FLAG') = 'Y''. To the left of the property name is a small icon representing a document with a green checkmark.

## **input\_type**

Specifies whether the weblet displays and receives time, date or datetime data.

If the weblet is to display and receive time data only, then you may prefer to use the Time (std\_time) weblet. If the weblet is to display and receive date data only, then you may prefer to use the Date (std\_date) weblet instead. Both of these weblets are a specialization of the Datetime weblet.

## **Default value**

'datetime'

## **Valid values**

'timeonly', 'dateonly' or 'datetime'. A list of the valid values may be displayed by clicking the corresponding dropdown button in the property sheet.

## **display\_in\_utc**

A Boolean property which determines if the datetime displays the datetime's UTC value or the datetime's local value.

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the DUTC attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## **allow\_sqlnull**

A Boolean property which determines if the datetime value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## date\_mask

Specifies the format or mask used to display and enter the date for the weblet. This is a string that contains a number of format specifiers that tell the weblet how to format the date.

Valid format specifiers are:

<b>Format Specifier</b>	<b>Description</b>
YYYY	Represents the year as a four-digit number.
YY	Represents the year as a two-digit number.
MMMM	Represents the name of the month as defined in std_script_messages.js
MM	Represents the month as a number from 01 through 12. A single-digit month is formatted with a leading zero.
M	Represents the month as a number from 1 through 12.
DDD	Represents the name of the day of the week as defined in std_script_messages.js
DD	Represents a day of the month from 1 - 31. A single digit day is formatted with a leading zero.
D	Represents a day of the month from 1 - 31.

**Note:** This specifies the presentation format the weblet uses. The input and output date received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

### Default value

'YYYY-MM-DD'

## **Valid values**

Any string containing valid format specifiers

## **time\_mask**

Specifies the format or mask used to display and enter the time for the weblet. This is a string containing a number of format specifiers that tell the weblet how to format the time.

Valid format specifiers are:

<b>Format Specifier</b>	<b>Description</b>
-------------------------	--------------------

HH	Represents the hour as a number from 0 through 23, that is, the hour as represented by a zero-based 24-hour clock that counts the hours since midnight. A single-digit hour is formatted with a leading zero.
H	Represents the hour as a number from 0 through 23, that is, the hour as represented by a zero-based 24-hour clock that counts the hours since midnight.
hh	Represents the hour as a number from 1 through 12, that is, the hour as represented by a 12-hour clock that counts the whole hours since midnight or noon. A single-digit hour is formatted with a leading zero.
h	Represents the hour as a number from 1 through 12, that is, the hour as represented by a 12-hour clock that counts the whole hours since midnight or noon.
mm	Represents the minute as a number from 0 through 59. A single-digit minute is formatted with a leading zero.
m	Represents the minute as a number from 0 through 59.
ss	Represents the seconds as a number from 00 through 59. A single-digit second is formatted with a leading zero.
s	Represents the seconds as a number from 00 through 59.
sss	Represents the milliseconds as a number from 000 through 999 All values are represented as three digits.
t	Represents A.M. or P.M.

**Note:** This specifies the presentation format the weblet uses. The input and output date received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

**Default value**

'HH:mm:ss'

**Valid values**

Any string containing valid format specifiers

**button\_image**

The path and file name, relative to the images virtual directory, of the image to display on the calendar prompt button.

**Default value**

'fp\_im003.gif' (this image is shipped with LANSA).

**Valid values**

The path and name of an image, relative to the images directory, enclosed in single quotes. An image can be chosen from a prompter by clicking the corresponding ellipses button in the property sheet.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

**button\_title**

Specifies text for the calendar button (if shown) that may display as tip text as the mouse moves over the button.

**Default value**

Blank – the text specified for the title property is used.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

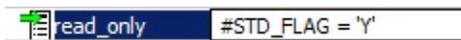
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor interface. On the left, there is a dropdown menu with 'read\_only' selected. To the right of the dropdown is a text input field containing the expression '#STD\_FLAG = 'Y''. The entire input area is enclosed in a thin border.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor interface, similar to the one above. The dropdown menu on the left still shows 'read\_only'. The text input field on the right now contains the expression 'key('field-value', 'STD\_FLAG') = 'Y''. The input area is enclosed in a thin border.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

## **hide\_calendar**

A boolean property, the result of which determines whether the calendar button is shown for the weblet.

### **Default value**

Blank – equivalent to false (that is, the calendar button will be shown if the weblet displays a date or datetime).

### **Valid values**

true(), false() or a valid expression.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

The width can also be affected by the size property, but in any event, the weblet will reserve a minimum width based on the data to be displayed.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).

## Time (std\_time)

### [QuickStart - Time Properties - Time](#)

The time weblet provides a text input box control with added features to support the display, entry and validation of times. It broadly corresponds to the `<input type="text">` HTML element.

An example of the weblet is shown below (for clarity it is shown with its label):

Sample time

The time weblet is best used with fields of type time. If you use this type, the data will automatically be passed in the format expected by the weblet. You can use the time weblet with fields of other numeric types such as packed or signed, but it is your responsibility to ensure the numeric value is formatted in the correct ISO format expected by the time weblet. For example, you could use a signed (6, 0) field containing a time in HHMMSS format with an edit word ('0 : : ') to format it as an ISO time format.

## QuickStart - Time

For time fields for which this weblet is the default visualization, you usually do not need to manually add it to your web page. Simply include your time fields in your web\_map or in a list that is present in your web\_map and they will be visualized using the time weblet. Similarly fields of type date and of type datetime will be visualized using the date (std\_date) and datetime (std\_datetime) weblets.

If you do need to add the time weblet to your page manually, simply drag the time field from the Fields tab onto your page. Alternatively, open the XSL for your webroutine in the LANSAs Editor and follow these steps:

1. Click on the Weblets tab, select Standard Field Visualization from the drop-down list near the top and locate the Time weblet.
2. Drag the weblet onto your page in the Design view. Click on the weblet and then click on the Details tab.
3. Set the name and value properties as required to associate the weblet with the required field in your webroutines web\_map.

## Properties - Time

The Time weblet's properties are:

allow_sqlnull	name	time_mask
class	onchange_script	title
disabled	pos_absolute	value
display_mode	read_only	width
hide_if	tab_index	

**name**

The name the weblet is identified with. If the weblet visualizes a field, this is the name of the field. Normally, you would leave this as the default and let LANSA use its own internal naming convention. However, you may want to use your own name if using JavaScript or XSL that references the weblet.

**Default value**

Where the weblet visualizes a field the default name is the field name or combines the field name with a row number (for fields in a list). Otherwise the default name is an automatically generated, unique identifier.

**Valid values**

Single-quoted text.

## value

The value to set the weblet to. If the weblet visualizes a field, this will identify the field whose value is to be shown.

## Default value

No default value applies – for most uses of this weblet you must specify a field whose value is to be represented by the input box and/or that is used to receive the contents of the input box.

## Valid values

Single-quoted text or the name of a field, system variable or multilingual text variable.

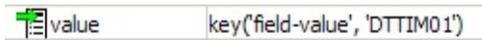
## Example

This shows how the value is specified when the weblet visualizes a field:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in blue, and its value is set to '#DTTIM01'.

When the property loses focus, the value is shown as follows:



A screenshot of a property editor for a weblet. The 'value' property is highlighted in blue, and its value is set to 'key('field-value', 'DTTIM01')'.

## **time\_mask**

Specifies the format or mask used to display and enter the time for the weblet. This is a string containing a number of format specifiers that tell the weblet how to format the date.

See the DateTime weblet for a full list of valid format specifiers.

**Note:** this specifies the presentation format the weblet uses. The input and output date received from and returned to the webroutine are always in ISO format. If you choose a different presentation format by setting this property, the weblet will convert to and from the internal representation as required.

### **Default value**

'HH:mm:ss'

### **Valid values**

Any string containing valid format specifiers.

## **display\_mode**

Controls whether the weblet accepts input, displays output or is hidden.

### **Default value**

Blank (equivalent to 'input').

### **Valid values**

Literal values 'input', 'output' or 'hidden'. A list of allowable values is available by clicking the corresponding dropdown button in the property sheet. Alternately, you may enter the name of a field, system variable or multilingual variable that will contain one of the allowable values at run-time.

## hide\_if

An expression which, if evaluated to be True, will hide the weblet.

### Default value

False() (that is, the weblet will always be shown)

### Valid values

Any valid XPath expression that returns a Boolean value.

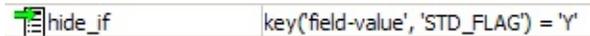
### Example

This example will hide the weblet if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a weblet property editor. The property name 'hide\_if' is highlighted in a dark blue box. To its right, the value '#STD\_FLAG = 'Y'' is entered in a text field.

When the property loses focus, the expression is shown as follows:



A screenshot of a weblet property editor. The property name 'hide\_if' is in a text field. To its right, the value 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in another text field.

## **allow\_sqlnull**

A Boolean property which determines if the time value can be left blank.

**Note:** This property must be consistent with the fields's repository definition (ASQN attribute).

### **Default value**

false(). If the weblet is dropped over a field, it defaults to the ASQN attribute of the field's repository definition.

### **Valid values**

true(), false() or a valid expression.

## **tab\_index**

Determines the tab order of the weblet on the form. The tab\_index property value determines the tab order as follows:

1. Objects with a positive tab\_index are selected in increasing tab\_index order (and in source order to resolve duplicates).
2. Objects with a tab\_index of zero or blank (the default) are selected in source order.
3. Objects with a negative tab\_index are omitted from the tabbing order. Note that this behavior is not defined in the HTML specifications and is only supported by Internet Explorer and Firefox.

### **Default value**

Blank. The weblet is selected in source order.

### **Valid values**

Blank or a valid numeric value.

**title**

Specifies text for the weblet that may display as tip text as the mouse moves over the weblet.

**Default value**

Blank – no tip text will be displayed.

**Valid values**

Single-quoted text or the name of a multilingual text variable (the corresponding ellipses button in the property sheet can be clicked to choose one from a list).

## **read\_only**

A Boolean property, the result of which determines whether the content of the weblet is read-only (that is, the user cannot modify the content).

### **Default value**

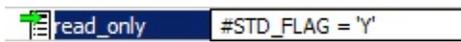
Blank – equivalent to False (that is, the user can modify the contents).

### **Valid values**

true(), false() or a valid expression.

### **Example**

This example will set the weblet to read-only if field #STD\_FLAG is equal to 'Y'. The expression should be entered in this form:



A screenshot of a property editor for the 'read\_only' property. The property name 'read\_only' is on the left, and the expression '#STD\_FLAG = 'Y'' is entered in the text field on the right.

When the property loses focus, the expression is shown as follows:



A screenshot of a property editor for the 'read\_only' property. The property name 'read\_only' is on the left, and the expression 'key('field-value', 'STD\_FLAG') = 'Y'' is entered in the text field on the right.

## **disabled**

A boolean property, the result of which determines whether the weblet appears enabled or disabled.

### **Default value**

Blank – equivalent to False (that is, the weblet will always be enabled).

### **Valid values**

true(), false() or a valid expression.

**class**

The Cascading Style Sheet (CSS) class name of the weblet.

**Default value**

The name of the shipped class for the weblet.

**Valid values**

Any valid class name from the Cascading Style Sheet, in single quotes. A list of available classes can be selected from by clicking the corresponding dropdown button in the property sheet.

## **pos\_absolute**

The absolute position of the weblet on the web page. Note that 'Position Absolutely' must be selected from the weblet's right-click menu for this property to be used. The property will usually be set in pixels by dragging and dropping the weblet.

### **Default value**

Blank (this is equivalent to the weblet being positioned relatively).

### **Valid values**

Valid 'left' and 'top' coordinates, in valid units of measurement, in single quotes.

## **width**

The width of the weblet on the web page. The weblet will reserve a minimum width based on the data to be displayed.

Usually you would set the width of the weblet by dragging the grab-handles around the weblet in the Design view of the LANSAs Editor. Doing so updates the value of the width property. However you can directly edit the property value if required.

### **Default value**

Blank (this is equivalent to the weblet adopting its default width).

### **Valid values**

A width, in a valid unit of measurement, in single quotes.

## **onchange\_script**

JavaScript code to be run when the input box loses focus after the text has been changed. JavaScript statements must be terminated by a semicolon.

### **Default value**

Blank. No JavaScript is run.

### **Valid values**

Any valid JavaScript statement(s).