# **Web Functions**

- Introduction to LANSA Web Functions
- Developing Applications with LANSA Web Functions
- Executing LANSA Web Function Applications
- WEBEVENT Functions
- LANSA Generated HTML Pages
- Default Process Pages
- RDML Tags
- Graphic Variables
- Web Components
- Web Function Editor
- Web Development Tips & Techniques
- Web Function & WEBEVENT Tutorials
- Appendix A. Header Styles
- Appendix B. LANSA for the Web XHTML

Edition Date November 8, 2013 © LANSA

### **1. Introduction to LANSA Web Functions**

As a LANSA developer, you have a choice of using Web Applications Modules (WAMs) or LANSA Web Functions.

Web functions were created as an extension to the original LANSA functions that are part of the core the LANSA development environment. Web functions allow you to deploy Web applications using HTML. For an introduction to LANSA Web Functions, review the following:

1.1 What are LANSA Web Functions?

1.2 Major Features of LANSA Web Functions

1.3 LANSA Web Function Architecture

1.4 What is the Web Functions Wizard?

1.5 Programming Language Skills

1.6 How Do You Develop Applications with LANSA Web Functions?

1.7 LANSA Web Function Transaction Server

For an outline of the implications of using Web Functions or WAMs, refer to 1.8 WEBEVENTs or WAMs? .

Note: Since Version 9.1, LANSA Web Functions generate XHTML meeting the 1.0 standards. The XHTML 1.0 standard and HTML 4.0 standards are almost identical. LANSA Web Functions generate HTML meeting the 4.0 standards. In this guide, the term HTML will be used described to generated HTML/XHTML. For details about XHTML, refer to LANSA for the Web XHTML.

### 1.1 What are LANSA Web Functions?

A LANSA Web Function is simply a LANSA function that has been Webenabled to allow access over the Internet. Web functions are procedural LANSA functions. They are part of a LANSA process. A Web-enabled function can be executed using a browser or using a 5250 terminal. Specialized WEBEVENT Functions are also procedural functions, but have been designed primarily for Internet access only.

By using LANSA Web Functions, you can build Web applications quickly and easily. Web Functions integrate seamlessly with the LANSA Application Development Environment (either Visual LANSA or LANSA for iSeries). You can build graphical HTML applications using the same Repository and RDML skill set used to build your iSeries and Windows applications. In fact, a LANSA program can be deployed to the iSeries, Windows or the Web using a single RDML program definition.

LANSA Web Functions allow you to build dynamic data retrieval applications for the Web. They also allow you to deploy the applications across a number of combinations of platforms including the iSeries, Windows, UNIX and Linux. Some of these platforms can be used as the Application/Data Server and/or the Web Server, which allows you to deploy your applications in a Multi-Tiered architecture.

LANSA Web Functions shield you from the technologies you have to learn to deploy Web-enabled applications. You do not need to learn code-intensive CGI or Java Servlets to build dynamic Web applications. You simply use LANSA's repository-based 4GL environment to build your application definition. LANSA allows the developer to work at the 4GL level. LANSA will automatically generate the HTML pages for you. LANSA Web Functions hide much of the complexity from the developer.

The LANSA Web Function architecture includes a Transaction Server that can provide your user sessions with a persistent connection between their browser and your Data Server. Persistent connections are not available in the HyperText Transfer Protocol (HTTP), the protocol used in serving the Web.

The huge growth and demand for Internet access to businesses has prompted LANSA to extend its offering beyond the traditional application development environment. LANSA offers a set of application specific frameworks to help companies deploy their e-business solutions at an even more astounding rate. Along with LANSA's powerful application development environment, Web developers will now benefit from Commerce Edition and the Web Functions Wizard to help accelerate Web application development.

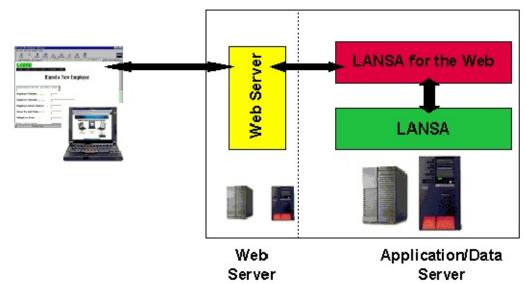
## **1.2 Major Features of LANSA Web Functions**

Some of the major features of LANSA Web functions include:

- Programmable templates to create LANSA applications for the Web.
- Seamless integration of CGI.
- Support for Java Servlets and IIS Plug-in (using ISAPI technology).
- Automatic generation of frame-based applications.
- Automatic generation of graphical HTML pages.
- Support for multiple clients, including Network Stations.
- Support of Multi-Tiered deployment architecture.
- Detailed knowledge of CGI, ISAPI or Java Servlets is not required.
- Transaction Server incorporated.

Note: Since Version 9.1, LANSA Web Functions generate XHTML meeting the 1.0 standards. The XHTML 1.0 standard and HTML 4.0 standards are almost identical. LANSA generates HTML meeting the 4.0 standards. In this guide, the term HTML will be used to describe the generated HTML/XHTML.

#### **1.3 LANSA Web Function Architecture**



The Web Server and Application/Data can be on a single platform (AS/400 or Windows), or they can be on multiple platforms.

A LANSA Web Function Application uses three basic software components:

- LANSA Development Environment (resides on the Application/Data Server)
- LANSA for the Web (resides on the Application/Data Server)
- Web Server (can reside on the same machine as the Application/Data Server, or can be a separate machine for multi-tier architecture).

The LANSA development environment has been used to build mission critical transaction based applications by thousands of companies worldwide. The LANSA Repository and 4GL technologies are proven technologies. They are reliable and productive. The LANSA Repository secures your data and maintains its integrity by enforcing centralized business rules. LANSA's 4GL (RDML) allows you to build applications quickly. Maintenance of LANSA applications is easy since working with a 4GL significantly reduces maintenance effort.

LANSA Web Functions are simply an extension of the LANSA development environment. LANSA's Web solution allows you to use your existing development skills. You still work with LANSA's Repository and 4GL (RDML) to develop and deploy your applications on the Internet. When compiling your RDML programs, you can simply indicate that the application will be enabled for the Web. LANSA's architecture allows you to Web enable existing LANSA applications by simply recompiling. LANSA protects your investment in your application systems. You can also develop new applications using WEBEVENT technology that are specific to the Web environment.

To complete the solution, you simply require TCP/IP and the Web serving software which provides the connection to the network

(Internet/Intranet/Extranet). The Web Server and Application/Data Server can be located on the same machine (using an iSeries) or you can use a multi-tier architecture where different machines are used for the Web Server and the Application/Data Server. For example, you could use a Linux or Windows Web Server with an iSeries Application/Data Server.

### 1.4 What is the Web Functions Wizard?

The Web Functions Wizard allows you to quickly and easily customize the presentation of your LANSA Web Function Applications. The Wizard is not used to create your LANSA functions. It is used to alter their appearance using Web-based functions executed from your browser. The Wizard removes the need to manually code HTML and JavaScript. It simplifies the implementation of graphic variables and Web components. You can change the look-and-feel of your applications without having to recompile any of the functions in your application.

The Web Functions Wizard allows the RDML programmer to concentrate on the functionality of the application, without having to worry about the presentation attributes. The presentation can be quickly assembled using simple menu choices and presentation options. The Wizard gives you the flexibility of deciding which components to include or exclude from the presentation.

The Web Functions Wizard removes the need for in-depth knowledge of HTML or JavaScript. The Wizard allows you to convert the default look-and-feel of a LANSA function into a contemporary Web look-and-feel. The pre-shipped images, color schemes and presentation schema reduce the need for specialized graphical skills as part of the application development and implementation.

For example, the screen shown is the default look-and-feel of applications generated by LANSA Web Functions.

<u>F</u> ile <u>E</u> dit <u>V</u> ie	ew F <u>a</u> vorites <u>T</u> o	ools <u>H</u> elp				
Hack For	· · ·	Refresh Home	Q Search	Favorites	3 History	» Lir
		Data Er	ntry			
mployee no						
Surname					것같다	
iven names						
ddress 1						
2월 33년 2월 2월 2월		88				
ddress 2	-		an and the state	배가 맛 다니다.		한 것 이 같은 것 같아요?
	a Distant (J. 1919) - Marcard A	in Particle Constraints and a second se				
Country	Product	Stock				
Country Product Code	Description		y all the second			
Product Code PCMOD001 IB		n <mark>Quantit</mark> 1	y all the second			
Product Code PCMOD001 IB PCMOD002 Hi PCMOD003 Do	Description M Aptiva ProLine ewlett-Packard Br ell Dimension XP:	N Quantit	y 0 5			
Product Code PCMOD001 IB PCMOD002 Hi PCMOD003 Do	Description M Aptiva ProLine ewlett-Packard Br	N Quantit	y 0 5			
Product Code PCMOD001 IB PCMOD002 Hi PCMOD003 Do	Description M Aptiva ProLine ewlett-Packard Br ell Dimension XP:	N Quantit	V 0 5 2			
Country Product Code CMOD001 IB CMOD002 Hi CMOD003 Di CMOD003 Ci	Description M Aptiva ProLine ewlett-Packard Br ell Dimension XP:	N Quantit	V 0 5 2			
Product Code PCMOD001 IB PCMOD002 Hi PCMOD003 Di PCMOD004 Ci	Description M Aptiva ProLine ewlett-Packard Br ell Dimension XP:	N Quantit 10 SB 5 8 B800R 1 330 5	V 0 5 2 4			
Country Product Code CMOD001 IB CMOD002 Hi CMOD003 Di CMOD003 Ci	Description M Aptiva ProLine ewlett-Packard Br ell Dimension XP:	N Quantit	0 5 2 4			

Using the Web Functions Wizard, you can convert this 5250 look-and-feel to a modern, state-of-the art look-and-feel as shown. There is no manual modification to the HTML and no recompilation of the function.

Data Entry - Microsoft Intern Eile Edit View Favorites	et Explorer Iools Help	
↔ ↔ ↔ ⊗ Back Forward Stop	Refresh Home Search Favoriles History Mail Print Ed: Discuss R	୍ର ଜୁନ Related
ddress 🙋 http://137.76.21.4:82	/CGI-BIN/LANSAWEB?WEBEVENT+L007BC83017E487005ACB094+WEB+ENG	💌 🤗 Go 🗌 Links
Lausa	Do you know these sites are built with LANSA ? www.fotbal.com Aspect Catalog Order Inquiry Online Shopping Home Pa	
Search GO! Layout Wizard Online Shopping Home Page Product Catalog	Data Entry Employee no Surname Given names Address 1 Address 2 Country	Announcements • Wireless accessibility for your AS/400 applications LANSA WML. • San Diego COMMON - March 12-17, 2000. LANSA Success Stories 1. Song File 2. The Riveria 3. Odessey Cruises 4. Sonus 5. Mikasa
PoweredBy	Product CodeProduct DescriptionStock QuantityPCMOD001IBM Aptiva ProLine10PCMOD002Hewlett-Packard Brio SB5PCMOD003Dell Dimension XPS B800R12PCMOD004Compaq Prosignia 3304	5. <u>Chown</u> 6. <u>Chown</u> Formore storits
	Next	The Local intranet

For more details, refer to the Web Functions Wizard Guide.

### 1.5 Programming Language Skills

By using LANSA Web Functions, developers apply their existing Repository and RDML skills to build Web-based applications. Although LANSA Web Functions can use CGI, Java Servlets or ISAPI technology, there is no requirement for a developer to have any knowledge of either of these interfaces.

There are NO new RDML commands introduced by LANSA Web Functions. The only programming skill required is RDML.

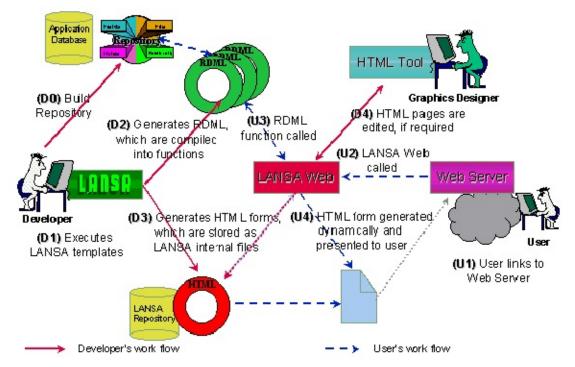
The HTML generation occurs when the LANSA process/function is compiled. The generated HTML is then stored as LANSA internal tables and not as static documents, as in conventional HTML programming.

The browser-based Web Functions Wizard allows developers to quickly and easily modify the user navigation, presentation and layout of LANSA functions without editing HTML or JavaScript.

You do not need to modify any of the LANSA generated HTML. However, if you want to modify the presentation of the HTML page, knowledge of HTML is essential. You will also need to understand the LANSA tags used in the HTML.

LANSA Web Functions use a number of JavaScript functions. If you need to use additional JavaScript functions in your applications, you will require a good knowledge of JavaScript. **Note:** If JavaScript support is disabled in your browser, you will not be able to execute any of these JavaScript functions.

### **1.6 How Do You Develop Applications with LANSA Web** Functions?

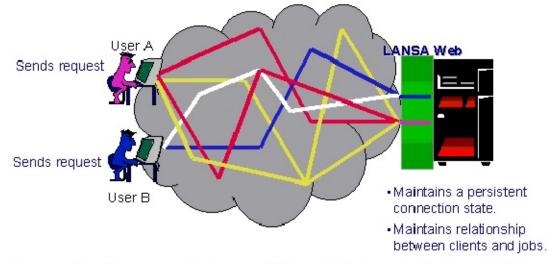


LANSA Web Function Applications are created using the same LANSA development environment used to create your iSeries or Windows applications. The development process is as follows:

- As with any LANSA application, the developer begins by populating the LANSA Repository. Fields and files are defined to create the application database.
- Next, the developer executes LANSA templates or manually codes RDML to create a set of functions. These programs might include a complex set of transactions, which inquire or update a set of database files.
- When the RDML function is compiled, an RPG or C/C++ program (depending upon the Application/Data Server being used) is created along with HTML forms, which are stored in internal LANSA files.
- Using the browser-based Web Functions Wizard, the user navigation, presentation and layout of the functions can be altered.
- Optionally, the LANSA HTML can be manually edited to enhance the graphical presentation. LANSA Web extensions such as Web Components may also be defined and used to enhance the HTML presentation.
- The developer's work is now complete.

At this point, the developer can have both a host version of the application and an HTML version of the application from the same set of code. When a browser requests a LANSA Web Function Application, the following occurs:

- A user links to the Web Server and requests a LANSA Web Function Application page.
- LANSA for the Web is called and manages the "application connection".
- The RDML function or program is executed on the Application/Data Server. The RDML function accesses the database.
- LANSA for the Web uses the stored forms to dynamically generate the HTML with the data required by the RDML function, and presents the information to the user.



#### **1.7 LANSA Web Function Transaction Server**



When executing applications over the Internet, the HyperText Transfer Protocol (HTTP) is used. HTTP does not support a persistent connection state. There is no concept of a user session. Each request is a new request, which is unrelated to any previous request. To overcome this limitation, LANSA for the Web provides a Transaction Server for Web Functions.

Note: The state management feature of the transaction server is only required if you have Web-enabled existing procedural applications which require session or state management. If you are developing applications specifically for the Web using WEBEVENT functions, this technology is not required.

The LANSA for the Web Transaction Server is used to maintain a persistent connection between the client devices and the Application/Data Server. The Transaction Server uniquely identifies each user and allocates a unique LANSA job for each user. Subsequent requests from users, which are known to the Transaction Server, are routed to the appropriate LANSA job which is active.

The Transaction Server maintains a connection state for each user and caters for unsynchronized page requests. An unsynchronized page request can happen as a result of the Back button supported in the browser paradigm. For example, the user can page back to previous functions and then request a page. This unsynchronized page request can be disallowed since the request is not the logical sequence expected by the active application on the Server. If an unsynchronized page request has been made, the user will be informed that the request is not valid and the current display page will be refreshed (shown) to the user.

LANSA allows you to write specialized Web Functions so that your typical Internet user can still use the Back button to navigate in your applications. This style of application is achieved by using the \*WEBEVENT option in your function. For more details, refer to WEBEVENT Functions.

The Transaction Server shields the developer from the complexities of maintaining connection states and allows you to concentrate on application design and development. The Transaction Server also includes a Transaction Monitor, which is used to monitor the activities of the LANSA jobs in process. LANSA for the Web allows you to specify a system wide timeout period or to have customized timeout periods for each user who is allowed to access your application.

The Transaction Server also allows you to limit the number of concurrent users on the Server. This feature allows you to restrict the number of active jobs running on your Server, thus limiting the amount of resources used by your Web applications.



WEB001 - Types of LANSA Web Functions

## **1.8 WEBEVENTs or WAMs?**

WEBEVENT functions and WAMs can be mixed in a single application.

If you already have applications developed using WEBEVENTs, those existing applications will continue to serve the purpose for which they were designed and you will be able to modify and enhance them into the future. WEBEVENT technology has been significantly extended with the addition of support for new data types, language syntax and constructs and the removal of significant limitations on field and list sizes.

However, a point may come in the lifecycle of your LANSA web applications when it is appropriate to begin using WAMs. You may wish to exploit WAMs in existing WEBEVENT applications in these circumstances:

- When you have a requirement to extend an existing web application beyond the browser perhaps to hand-held devices. Of course, this is a business requirement, which may necessitate adopting WAM technology.
- When you are extending an existing web application with a significant number of discrete "chunks" of new functionality. You might develop these new parts using WAMs.
- When making major revisions driven by business requirements that will substantially impact large parts of your application. Depending on the extent of the impact, you might decide that this is an opportune time to "future-proof" your web applications with WAMs.

Some of the implementation details of WEBEVENT and WAM applications are different and there are certain considerations and certain techniques you need to adopt. Refer to WAM and WEBEVENT Interoperability Techniques for details.

In general, you should use WAMs for new web applications.

## Why did LANSA develop WAM technology?

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 more readily adaptable to new technologies as they emerge.

2. It is clear that application development is heading towards a componentbased future and LANSA offers support for component-based development. WEBEVENT are functions and their architecture is not best-suited to a component-based world. For this reason, it is imperative that LANSA allows you to take full advantage of component-based techniques.

### 2. Developing Applications with LANSA Web Functions

If you are developing a LANSA Web Function Application for the very first time, you should review the following:

- 2.1 Before You Begin Checklist
- 2.2 Web Enabling a LANSA Process
- 2.3 Types of Web Functions
- 2.4 Example of a Procedural Function
- 2.5 Example of a WEBEVENT Function
- 2.6 Shipped LANSA Web Function Templates
- 2.7 Compiling Functions
- 2.8 Using the e-Business Framework Wizard

For an introduction to executing LANSA Web Function Applications, you should review the following:

2.9 Calling LANSA Web Processes and Functions

2.10 Executing Applications: Process Menu

2.11 Executing Applications: WEBEVENT

For more details about executing LANSA Web Function Applications, refer to Executing LANSA Web Function Applications.

### 2.1 Before You Begin Checklist

• Before you begin your development efforts, you should have a properly installed and configured LANSA for the Web system. You will need a properly configured Web Server and a properly configured Application/Data Server. For details about the installation and configuration of LANSA for the Web, refer to the Installing LANSA on Windows Guide and the Installing LANSA on iSeries Guide.



If you are using LANSA Web functions for the very first time, it is recommended that you complete the Tutorials.

### 2.2 Web Enabling a LANSA Process

A LANSA process is like a menu as it acts as the parent or container for the functions in your Web application. In order to execute your LANSA functions using the Web, the functions must be part of a Web or XML enabled LANSA process.

Once you have created a LANSA process, you must then Web enable it. To do this:

a.Select the Process in the *Repository* tab.

📜 LANSA Editor - PSLSYS ( P	rocess ) - Per	sonne	l Syst	em Main	Menu					
<u>File E</u> dit <u>V</u> iew <u>O</u> ptions Ve <u>r</u> ify	We <u>b</u> _Debug	<u>T</u> ools	<u>W</u> indov	v <u>H</u> elp						
📗 🗟 New 🔻 🔌 💾 💩	X 🗗 🖻	う	6	0 🛛		<u>s</u>   Q		<u>ه</u> •		
Details		Func	tions	Definition	Atta	chments	Specia	al Entrie	es Parame	te < >
PSLSYS	~			-:!-						
Menu style SAA/CUA Action bar Anticipated usage Light Heavy			Men Anti Opti Enal Gen	u style cipated us mize for re ble for we erate XML <b>Itilingual</b>	emote o b		ations	Yes Yes	JA nel System	Main Me
<ul> <li>Optimize for remote communicati</li> <li>Enable for web</li> <li>Generate XML</li> </ul>	ions	<	Frar	içais				Menu p	orincipal de (	gestion (
Description(s) English	sonnel System u principal de g	× =								
8	🁉 📚 🏠 🐥		Assista	nt						
Ready				LAI	VSA1	DEM	PCXU:	SER	PCXTASK	ENG

b.Select the *Definition* tab.

c.Select *Details* in the *Definition* tab or select the *Details* tab.

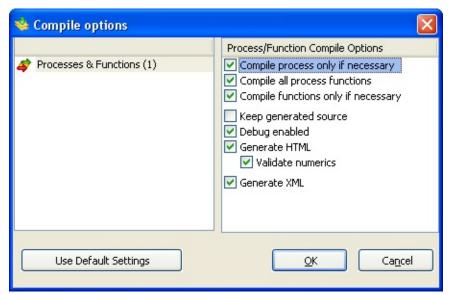
d.In the *Details* tab, select (tick) the *Enable for Web* and/or *Enable for XML* options.

The *Enable for the Web* option allows you to generate HTML. The *Enable for* 

*XML* option allows you to generate XML. You may use one or both of these options.

**Note:** XML support for web functions should only be used for preexisting XML-enabled processes. For new development, please consider using WAMs.

Once these options are selected, subsequent compiles of the functions in the selected LANSA process will automatically generate the required HTML and/or XML.



The generated forms are stored in the LANSA internal tables. You can edit these forms by using the Web Function Editor.

If you have existing LANSA processes, you can simply Web or XML enable them and then recompile the functions to generate the required forms.

### 2.3 Types of Web Functions

With LANSA for the Web, you can deploy two types of functions:

- Procedural Functions
- WEBEVENT Functions

### **Procedural Functions**

LANSA allows you to Web enable your existing procedural LANSA applications for execution over the Web. LANSA provides a transaction server to maintain the state of the functions as they execute. For details, refer to LANSA Web Function Transaction Server.

For example, the Personnel Demonstration system executes on 5250 terminals using an iSeries and executes under Windows on a PC. This application was written as a procedural function in 1987. This application was written as a procedural function in 1987. The Personnel Demonstration system can be Webenabled and recompiled, and then it can be executed over the Internet. Refer to 2.10 Executing Applications: Process Menu. and you can try it out for yourself. The Personnel System has been web enabled for you and you can try it out for yourself. See **Note** following.

When a procedural function executes over the Web, the user cannot use the browser's Back button. In order to navigate within the application, the user must use the Cancel button (which is equivalent to an F12=Cancel request) which is automatically included into the application by LANSA for the Web. The application executes in a procedural mode just like it would using a 5250 terminal on the iSeries.

All procedural functions that will run on the web should have the EXIT\_KEY enabled. Having the EXIT\_KEY disabled could result in unpredictable behavior. For example, a function might never timeout or, in some circumstances, use an abnormally large amount of CPU time.

### WEBEVENT Functions

LANSA allows you to code functions specifically for the Internet user and the browser paradigm. In the browser paradigm, a persistent connection is not required. Each page is independent. You cannot dictate the navigation path of your application to the user. The Internet user has a Back button in the browser which allows them to go back several pages and resubmit a request.

A function is defined to be a WEBEVENT function by specifying the following

option in the FUNCTION command:

#### **FUNCTION OPTIONS(\*WEBEVENT)**

With WEBEVENT functions, the LANSA function will terminate after a DISPLAY or REQUEST command is executed. No other logic will be executed after the HTML page for the function is sent. Using this technique, you must code your functions to properly process the information when the user decides to submit a screen with data. The logic of the RDML function must be written specifically for WEBEVENT. You cannot simply add the \*WEBEVENT keyword to your functions.

The majority of your newly created Web applications will be written as WEBEVENT functions.

For more details, refer to **WEBEVENT** Functions.

### WEB001 - Types of LANSA Web Functions

**Note:** The **LANSA Personnel Demonstration System** was originally created in 1987 to execute on a System 38. This same application can still be executed on IBM i as well as Windows, WEB and Linux platforms. The original database and application are virtually unchanged. There have been some extensions to the database to demonstrate RDMLX concepts.

There are two web versions of the Personnel System, one uses WEBEVENT technology and the other uses WAMs. These are included with the IBM i demonstration material and can be checked out to a Visual LANSA environment as required. These applications can be used to give you ideas of how to create and include the elements required for both WEBEVENT and WAM processing.

**webevent** (web enabled RDML or RDMLX partition): http://<web server>/cgi-bin/lansaweb? procfun+lansadem+ldem+dem

wam (web enabled RDMLX partition, only supported for technology
service LANSA:XHTML):
http://<web server>/CGI-BIN/lansaweb?
webapp=LWAMDEM+webrtn=ldemhome+ml=LANSA:XHTML+

### 2.4 Example of a Procedural Function

A simple header / detail function is a good example of a procedural function which requires the transaction server. If you execute the FRENQ02 template, you can create a procedural function which requests search information and then displays the detailed results.

The overall structure of the RDML in the function might appears as follows:

FUNCTION	OPTIONS(*DIRECT)
GROUP_BY	NAME(#HEADER)
DEF_LIST	LIST(#LIST)
BEGIN_LOOP	
REQUEST	FIELDS(#DEPTMENT)
FETCH	FIELDS(#HEADER)
SELECT	FIELDS(#LIST)
ADD_ENTRY	TO_LIST(#LIST)
ENDSELECT	
DISPLAY	FIELDS(#HEADER) BROWSELIST(#LIST)
•••	

END\_LOOP

When the function encounters the REQUEST statement, a page is sent to client browser and the function must wait for a response from the user. The function is still active and the transaction server maintains the state of the function. When the user responds, the function continues until the DISPLAY statement when a page of search results is sent back to the client browser. The function is still active and the transaction server continues to maintain the state.

If the browser's Back button is used to return to the REQUEST page, the transaction server will detect the sequence error when the REQUEST page is submitted to the Web Server again. The transaction server knows that the LANSA function is expecting input from the DISPLAY statement and not the REQUEST statement. The DISPLAY page will be sent to the client browser again with an error message.

With Web applications, you may wish to allow the user to go back and change the requested information using the browser's Back button. WEBEVENT functions are designed specifically for this type of functionality.



### 2.5 Example of a WEBEVENT Function

The simple header / detail function can also be written to execute as a Web event function. If you execute the FRWEBENQ01 template, you can create a WEBEVENT function which request search information and then display the detailed results. The function created by FRWEBENQ01 is a re-entrant WEBEVENT function. For more details about re-entrant functions, refer to Handling Re-entrant Functions.

The overall structure of the RDML in the function will appear very different to the procedural example. The function will contain a REQUEST for the search data and a DISPLAY of the search results. However, the location of the processing logic is very different in the WEBEVENT function. WEBEVENT functions terminate immediately after processing a REQUEST or DISPLAY statement.

When the WEBEVENT function encounters the REQUEST statement, a page is sent to the client's browser and the function terminates. There is no need to maintain the state, as the LANSA function is no longer active.

When the user responds, the function must execute again because it has terminated. The function receives the information from the REQUEST and will select the required data to execute the DISPLAY statement. Again, once the DISPLAY is executed, the function terminates.

If the browser's Back button is used to return to the REQUEST page, there is no problem. When the user submits the REQUEST again, the function simply executes again. It receives the information from the REQUEST and will select the required data to execute the DISPLAY statement as before.

LANSA automatically handles the exchange of the data. The developer must properly code the RDML in the function knowing that a function terminates once a REQUEST or DISPLAY statement is encountered. The function is written with the processing logic always preceding the screen display.

For more detailed examples, refer to **WEBEVENT** Functions.



WEB001 - Types of LANSA Web Functions

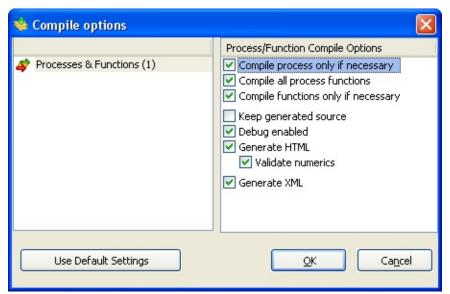
### **2.6 Shipped LANSA Web Function Templates**

- LANSA for the Web ships sample templates for generating WEBEVENT functions:
- FRWEBADD01 Data entry for \*WEBEVENT.
- FRWEBENQ01 Page at a time with detail display.

Template FRWEBADD01 creates a simple data entry WEBEVENT function. Template FRWEBENQ01 creates a WEBEVENT function that has a generic search which displays the results of the search in a page at a time browse list. The user can select any of the browse list entries for a more detailed display of information.

## **2.7 Compiling Functions**

If a LANSA process has the appropriate flags set, LANSA will generate the HTML and/or XML pages for the functions when the functions are compiled. (Refer to 2.2 Web Enabling a LANSA Process.) The compile will also create the RPG or C/C++ program objects which provide the application programming logic on the Application/Data Server.



If for some reason, you do not want the HTML or XML pages generated, you can deselect the Generate options before compiling the function. Note that if you do this, when a LANSA Process is compiled, there is no generation of HTML or XML.

The LANSA Web function pages are stored in LANSA internal files. This allows for easier backup/restore and maintenance procedures. These pages cannot be accessed without going through LANSA, thus preventing unauthorized editing. You can access these pages by using the Web Function Editor.

When compiling WEBEVENT functions, warning messages may appear as the function logic is checked to ensure that it will execute properly as a WEBEVENT function. For more details, refer to Considerations for WEBEVENT Functions.



### 2.8 Using the e-Business Framework Wizard

Once you have created your LANSA Web processes and functions, you can use the e-Business Framework Wizard to customize the user navigation, presentation and layout of your functions. The e-Business Framework Wizard allows you to use browser-based functions to customize the presentation of your HTML pages used in your LANSA Web function applications. The e-Business Framework Wizard is primarily used with WEBEVENT applications.

Advanced Software Made Simple	e-Business Framework
	Welcome to the e-Business Framework Wizard
	Use this wizard to customize the presentation of your LANSA applications.
Home Define Process Group	<ul> <li>The DEFAULT Process modifies your system defaults for browse list, layout, left menu, horizontal menu, right menu and style sheet. Modify DEFAULT values with care as the changes may impact more than your current application.</li> </ul>
File Search Help	<ul> <li>To work with a subset of processes which comprise your LANSA for the Web application, use the "Define Process Group" to nominate a generic start for the group of processes which comprise your application.</li> </ul>
Language: English 💌	<ul> <li>All browse lists from subset of processes are available when the browse list option is selected. The DEFAULT browse list relates to the default system variables used for browse list presentation is no specific browse list values have been defined. (This is particularily useful for browse list standardization across your application).</li> </ul>
Characteristics to modify: Layout	<ul> <li>To get started simply select the language you want your HTML pages generated in (if you are working in a multilingual system), select the web page characteristics to be modified or reviewed, select a process to review (unless working with browse lists or system variables) and then GO!</li> </ul>
All Processes:	
60	

The e-Business Framework Wizard allows you to:

- Change the value of LANSA graphic variables.
- Change the presentation of your LANSA browse lists.
- Change the layout of your application.
- Customize Menu components.
- Adopt layouts.

The e-Business Framework Wizard simplifies the development of your Web function applications. It uses sentence like-descriptions for the variables and hides their names from the developers. Components are built using questions and answers instead of manually coding HTML or JavaScript.

For more details about the e-Business Framework Wizard, refer to the Introduction to the e-Business Framework Wizard in the *Web Functions Wizard Guide*.

### 2.9 Calling LANSA Web Processes and Functions

For complete details about calling LANSA processes and functions, refer to Executing LANSA Web Function Applications

Once you have compiled the functions in your Web-enabled process, you are ready to call or execute the application over the Web. If you are using a procedural function, you can call the LANSA process or you can call the LANSA function directly. If you are using a WEBEVENT function, you must call the LANSA function directly.

LANSA for the Web uses URLs to call your LANSA Web function applications. The URL to call your LANSA applications involves invoking the LANSAWEB or LANSAXML program. The URL syntax to call your Web-enabled HTML application is:

```
http://<web server>:port/CGI-BIN/LANSAWEB?<parameters>
and for XML it is:
```

```
http://<web server>:port/CGI-BIN/LANSAXML?<parameters>
```

### **Calling a LANSA Process**

The parameters to call to a LANSA process are:

#### PROCESS+<process name>+<partition>+<language>

where, partition and language are optional.

For example, if you want to create a link to the PSLSYS process in the DEM partition, your URL might be entered as:

#### http://www.lansa.com/CGI-BIN/LANSAWEB? PROCESS+PSLSYS+DEM+ENG

### **Calling a LANSA Function Directly**

The parameters to call a LANSA function are:

#### PROCFUN+<process name>+<function name>+[<partition>]+ [<language>]

For example, if you want to execute the ENROL function in the PSLSYS process in the DEM partition, your URL might be entered as:

http://www.lansa.com/CGI-BIN/LANSAWEB? PROCFUN+PSLSYS+ENROL+DEM+ENG WEBEVENT functions must always be called directly.

WEB001 - Types of LANSA Web Functions

### 2.10 Executing Applications: Process Menu

When executing a LANSA process using LANSA for the Web, the application has a frameset look and feel. The frameset style divides the working area of the browser into two areas - the Menu area and the Client (body) area.



You can choose to have the frameset style persistent by selecting the "Always show frames for CUA/SAA style processes" option in the LANSA for the Web Administrator.

Once a LANSA process has been called, the user can execute any of the functions listed in the menu. These functions cannot be WEBEVENT functions. These functions should only be procedural functions. Procedural functions have their state managed by the LANSA Web Transaction Server and the user cannot use the browser's Back button.

If the frameset style is not persistent selecting a menu item which calls a LANSA function will refresh the display of your browser. The menu frameset will not be displayed while the browser is displaying a LANSA function.

🖞 Enral a New Employee - Micro	soft Internet Explorer	
<u>Eila E</u> dit ⊻ew <u>G</u> o F <u>a</u> voritea		æ
	44 D 7 3 5 5	links
HOME IN MENU IN NEXT IN BACK	SEARCH HELP	
	Enrol a New Employee	
Employee Number		
Employee Surname		
Employee Given Name(s)		
Street No and Name		i ferrialiant Carl Antonio
Suburb or Town		
State and Country		
Post / Zip Code		
성감 아님께서, 전에서 나라면서,		
Home Phone Number	한테 이 것은 것 같은	
Business Phone Number		

If you were to call the procedural function directly, it would appear exactly as shown when called from a process menu.

LANSA Web functions have been designed to allow all the generated HTML pages to have a consistent look and feel. By default, all HTML pages generated will have the same Menu Area background for all LANSA processes and the same Client Area background for all LANSA functions. Each LANSA function will have the same set of images displayed in the same positions in each of their respective HTML pages. The same set of images is displayed at the top (standard header) and the bottom (standard footer) of the HTML pages.

LANSA Web functions have a default set of image settings to be used with your pages. You can override these defaults if you wish, or you can specify additional settings which will be used by your LANSA Web functions.

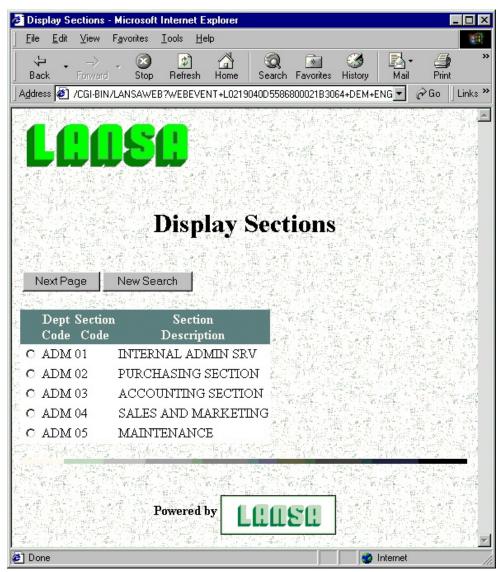
For details about how to call a LANSA process, refer to 2.9 Calling LANSA Web Processes and Functions.



## 2.11 Executing Applications: WEBEVENT

WEBEVENT functions must always be called directly. They cannot be called from a LANSA process menu.

When executing a LANSA function directly using LANSA for the Web, the application does not use a frameset.



When a user invokes a LANSA function, it will refresh the display of your browser. Its presentation is based on the layout of the function.

LANSA Web functions have been designed to allow all the generated HTML pages to have a consistent look and feel. Each LANSA function will have the same set of images displayed in the same positions in each of their respective HTML pages. The same set of images is displayed at the top (standard header)

and the bottom (standard footer) of the HTML pages. However, the default user buttons for navigation in procedural functions will not be displayed. The user navigation is controlled by developer defined buttons or links in the function. For details about how to call a LANSA function directly, refer to 2.9 Calling LANSA Web Processes and Functions.



# **3. Executing LANSA Web Function Applications**

LANSA Web Function Applications are generally hosted from your corporate Web site. In order to set up your **Web** site and execute your LANSA Web Function Applications, you should review the following:

3.1 No Existing Web Site

- 3.2 Integrating with an Existing Web Site
- 3.3 Before You Deploy Your Applications
- 3.4 Uniform Resource Locator (URL) Syntax
- 3.5 Calling a LANSA Process
- 3.6 Calling a LANSA Function Directly
- 3.7 Passing Parameters to a LANSA Function
- 3.9 Debugging using iSeries Batch Debug
- 3.10 Debugging using Visual LANSA
- 3.11 Invoking the iSeries Spool File Facilities
- 3.12 Event Logging
- 3.13 Enabling Event Logging
- 3.14 Logging User Defined Fields

## 3.1 No Existing Web Site

If you do not have an existing Web site, you will need to obtain a registered Internet domain name and an Internet Protocol (IP) address for your site. These can be arranged with any Internet Service Provider (ISP). Your ISP will be able to advise you on the best way of attaching your Web Server on the Internet.

Once you have attached your Web Server on the Internet and have set up your Web site, you will need to have a home page for your Web site.

To create your own home page, you can use a number of Web tools such as Microsoft Front Page. Your home page can be created using a simple text editor to enter your HTML.

Once you have a home page created, save the file as INDEX.HTM. (If you have changed the name of the index page in the Web Server configuration, this file should be saved to the new name instead. By default, the Web Server will use the INDEX.HTM page.)

Copy the INDEX.HTM file to the home directory of your Web Server.

After you have set up your Web site and the home page, you can then deploy your Web enabled LANSA applications on the Internet by following the instructions in 3.2 Integrating with an Existing Web Site.

## 3.2 Integrating with an Existing Web Site

If you already have an existing Web site, you can quickly integrate your LANSA applications into your Web site by simply including a link to the LANSA application from your selected HTML page.

The syntax of the Uniform Resource Locator (URL) to link to your LANSA applications is described in the 3.4 Uniform Resource Locator (URL) Syntax section of this guide. There is no limit to the number of links you can create. You can link to a LANSA process or directly to a LANSA function.

To use Multi-Tier deployment with a Web Server other than an iSeries, you must use LANSA for the Web Java Servlet Support or LANSA for the Web IIS-Plug-In.

Your existing Web site does not have to be running on an iSeries. Your existing Web Server can be running on any platform. The important thing is that you have a Data\Application Server that caters for your LANSA application. The URL is used to link your Web-enabled LANSA applications to your existing Web site.

## **3.3 Before You Deploy Your Applications**

At this stage, it is assumed that you have installed and have correctly configured the Web Server.

Once your LANSA processes and functions have been compiled successfully, you are ready to test them. You will need a browser capable of supporting HTML V3.2 or later. The browser must be capable of supporting the use of frames as well as JavaScript.

Before you test your applications, you may want to review the Web Server and Data/Application Server settings using the LANSA for the Web Administrator.

## 3.4 Uniform Resource Locator (URL) Syntax

Uniform Resource Locators provide a standard method of identifying resources that are available using Internet protocols. LANSA for the Web uses URLs to launch your LANSA Web Function Applications.

It is recommended that you create an HTML page that contains the URLs of all your LANSA applications. This page can also be used as the Home Page of your Web site.

The URL to call your LANSA applications involve invoking the LANSAWEB or LANSAXML program, with appropriate parameters. By default, the LANSAWEB/LANSAXML program resides in the CGI-BIN library defined in your Web Server.

The URL syntax to call your Web-enabled HTML application is:

http://<web\_server>:<port>/CGI-BIN/LANSAWEB?<parameters>
and for XML it is:

http://<web\_server>:<port>/CGI-BIN/LANSAXML?<parameters>
where:

<web\_server> is the name of your Web Server, as registered to a Domain Name Server(DNS);

<port> is the port identifier defined for your LANSA system. This parameter is
optional if your LANSA system is assigned the default port 80;

<parameters> is a list of parameters passed to the LANSAWEB/LANSAXML
program. (Refer to 3.5 Calling a LANSA Process, 3.6 Calling a LANSA
Function Directly and 3.7 Passing Parameters to a LANSA Function.)

Note: the URL sub-path "/CGI-BIN/LANSAWEB?" and "/CGI-BIN/LANSAXML?" MUST be in upper case if you are using Java Servlet.

# **3.5 Calling a LANSA Process**

You can link a call to a LANSA process using the following parameters in your call to the LANSAWEB/LANSAXML program.

## PROCESS+<process\_name>+<partition>+<language>

where:

PROCESS (can be in lower-case) is the keyword instruction to LANSA for the Web to execute a LANSA process;

<process\_name> is the name of the LANSA process you want to execute;

<partition> is the LANSA partition. This parameter is optional. If this
parameter is not specified, the default LANSA partition will be used;

<language> is the partition language you wish to use. This parameter is also optional. If a partition language is not specified, the default partition language will be used.

If you use the <language> parameter, the <partition> parameter must also be specified, otherwise, the <language> parameter is assumed to be the <partition> parameter.

Note that each parameter is delimited by the plus (+) character as in this example:

#### http://<web\_server>/cgi-bin/lansaweb?process+<process\_name>+ <partition>

As an example, to create a link to a web system showing LANSA's web-enabled Personnel Demonstration, which shows a 5250 (green screen) application, your URL might appear as follows:

#### http://<web\_server>/cgi-bin/lansaweb?process+pslsys+<partition>

To execute an alternative link to the Personnel Demonstration, using the process specifically designed as a WEB interface, your URL might appear like this:

### http://<web\_server>/cgi-bin/lansaweb?process+lansadem+ <partition>

For XML, your link would be like this:

http://<web\_server>/cgi-bin/lansaxml?process+pslsys+<partition>

To call a LANSA function directly, refer to 3.6 Calling a LANSA Function Directly.

# 3.6 Calling a LANSA Function Directly

You can execute a LANSA function directly using the following parameters in your call to the LANSAWEB/LANSAXML program.

# PROCFUN+<process\_name>+<function\_name>+<partition>+<language>

where:

PROCFUN (can be in lower-case) is the keyword instruction to LANSA for the Web to execute a LANSA function directly;

<process\_name> is the name of the LANSA process containing the function;

<function\_name> is the name of the LANSA function you want to execute;

<partition> is the LANSA partition. This parameter is optional. If this
parameter is not specified, the default LANSA partition will be used;

<language> is the partition language you wish to use. This parameter is also optional. If a partition language is not specified, the default partition language will be used.

If you use the language parameter, the <partition> parameter must also be specified, otherwise, the <language> parameter will be interpreted as the <partition> parameter.

Note that each parameter is delimited by the plus (+) character as in this example:

# http://<web\_server>/cgi-bin/lansaweb?procfun+<process\_name>+<function\_name>+<partition>

As an example, to create a link to a web system showing LANSA's web-enabled Personnel Demonstration, which shows a 5250 (green screen) application, your URL might appear as follows:

#### http://<web\_server>/cgi-bin/lansaweb?procfun+pslsys+enrol+ <partition>

To create a link to the same Personnel Demonstration using the process specifically designed as a WEB interface, your URL might appear like this:

#### http://<web\_server>/cgi-bin/lansaweb?procfun+lansadem+ldem+ <partition>

For passing parameters when calling a LANSA function, refer to 3.7 Passing Parameters to a LANSA Function.

## **3.7 Passing Parameters to a LANSA Function**

You can pass parameters to a LANSA function that is executed directly.

If you want to pass parameters to a LANSA function, you would extend the URL of calling a LANSA function directly to include:

#### +FUNCPARMS+param1+...+param5

The keyword, FUNCPARMS (can be in lower-case), is used to indicate that the rest of the URL are the parameters to the function. Each parameter must follow the syntax:

#### <field\_name>(tllld):value

where

<**field\_name**> is the name of the field to receive the parameter

Τ	is the field type (A for alphanumeric, P for packed and S for signed, L for lowercase alpha fields)
Lll	is the length of the field value with leading zeros
D	is the number of decimal positions
Value	is the value of the parameter.

For example, if you wanted to pass a value of 'A0001' to the EMPNO field, the syntax of the parameter would be:

### EMPNO(A0050):A0001

Your final URL might appear as follows:

#### http://www.lansa.com/cgi-bin/lansaweb? PROCFUN+PSLSYS+ENROL+WEB+FUNCPARMS+EMPNO(A0050):A0(

Note that the length of the field includes leading zeros and the delimiting character used is the colon (:) character.

If the value of the parameter includes embedded blanks, you must surround the value with double quote (") characters.

You are allowed to specify up to 20 function parameters in a URL.

For a method of passing parameters, you may also wish to refer to HTTP Header Variables in the *Installing LANSA on Windows Guide*.

# 3.8 Specifying a Task Identifier

If your LANSA partition is enabled for Task Tracking, you will need to specify a task to run your LANSA function. To use task tracking, extend the URL syntax discussed above to include:

#### +TASK\_ID+<task\_identifier>

where <task\_identifier> is a valid task defined for the LANSA partition.

Note that if your LANSA partition is enabled for Task Tracking and you want to run the Layout Wizard in this partition, a task identifier must be specified in the URL.

# 3.9 Debugging using iSeries Batch Debug

LANSA for the Web allows your RDML functions to be debugged in batch on the iSeries. This feature requires the use of an iSeries display device that is not currently signed on.

To invoke batch debugging of your RDML functions, you would extend the URL syntax discussed above to include:

#### +BDEBUG+<device>+<message\_queue>

where

<device> is the name of the iSeries display device;

<message\_queue> is the name of the message queue you wish to use.

The <device> and <message\_queue> parameters are optional. If you do not specify a display device, the debugging information will be directed to any active display device on your network.

Your LANSA functions must be compiled for debugging purposes to use this option.

For example, if you want to debug the ENROL function in the PSLSYS process in the WEB partition, your URL might appear as follows:

#### http://www.lansa.com/cgi-bin/lansaweb? PROCFUN+PSLSYS+ENROL+WEB+BDEBUG+QPADEV0001

Note: It is very important that the device specified is not allocated. Make sure that no one is signed on to the device.

#### Also see

3.10 Debugging using Visual LANSA

# 3.10 Debugging using Visual LANSA

For set up and a detailed description of debugging using the Visual LANSA development environment, please refer to the Interactive Debugging in the *LANSA for Web Housekeeping Guide*.

In addition, when you compile your LANSA functions, you must select the *Debug enabled* option as described in Compile and run the Program in the *Developers Guide* in order to build binaries with debugging capability.

In the following description, Debug Host or Debugger computer is referring to the computer where a Visual LANSA development environment is running and to be used for interactive debugging. That is not the Data/Application Server where the Web Functions run, unless of course you have both the Visual LANSA and the Data/Application Server running on the same machine.

#### Also see

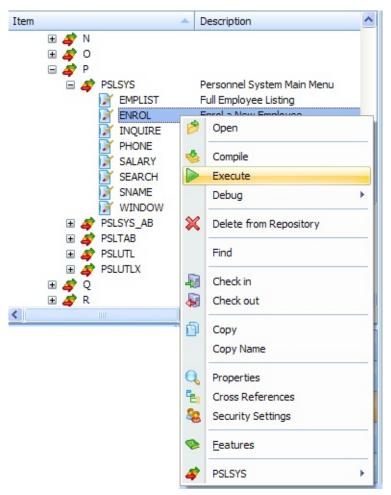
3.10.1 Start Development Environment Debugging Session

3.10.2 Start Web Browser Debugging Session

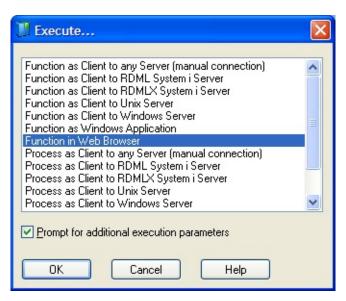
3.9 Debugging using iSeries Batch Debug

## 3.10.1 Start Development Environment Debugging Session

In the *Repository* tab of the *LANSA Editor*, locate the **Web** function to be debugged. Right click on it to open the context menu. Select *Execute* from the context menu.

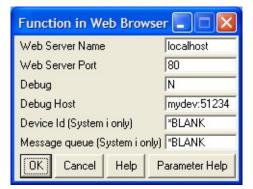


The Execute dialog will be displayed:



Select either *Function in Web Browser* or *Process in Web Browser* to execute the selected Web function.

Select (check) the option *Prompt for additional execution parameters* to open the following dialog where you can specify that you want Debugging to be activated. If the *Prompt for additional execution parameters* option is not selected, the function or process will be run without any debugging session.



Check that the *Web Server Name* and *Web Server Port* are correct for your installation.

Change the *Debug* parameter from N to Y to start the Web function for debugging.

For IBM i batch debug, you need to specify the *Device Id* and *Message Queue*. (IBM i Batch Debug is not done within the Visual LANSA Development Environment.)

*Debug Host* is set by default according to LANSA Debug Settings (see Debug in LANSA Settings in the *User Guide*) and refers to the Visual LANSA development environment that you want to use for interactive debugging.

Normally, it refers to the current development environment you are using. If you want to temporarily use a different development environment (which can be on a different computer) for the debugging session, you may change Debug Host to a value such as mydevenv:51234. This is name of the computer with the development environment that you want to use and the network port number used by the Visual LANSA Debug Service of that development environment. You will find this information in the *Debug* dialog of the *LANSA Settings* of the development environment you want to use for debugging.

The LANSA Settings		
	Debug	<u>o</u> k
General	Debugger Location	<u>C</u> ancel
1	Computer Name	Apply
Source	Debug Service	Reset
	Port 51234	Reset All
Design XSL Oebug	<ul> <li>Start debug service with LANSA Editor</li> <li>Restart service</li> <li>Debug service is started and running with port 51234</li> <li>Break at first executable statement</li> </ul>	Reset Editor

The parameters *Device Id* and *Message Queue* are not used for Visual LANSA interactive debugging.

## 3.10.2 Start Web Browser Debugging Session

To start interactive debugging for your RDML (Windows only) or RDMLX functions from a web browser, extend the URL syntax to include:

#### +BDEBUG+REMOTE=<debug\_host>

where <debug\_host> consists of the name of the computer where the development environment that you want to use for debugging is running and the network port number used by the Visual LANSA Debug Service of that development environment and the 2 pieces of information are separated with a colon ':', for example, mydevenv:51234. You can find both information from the LANSA Settings dialog of the development environment you want to use for debugging. (See LANSA Settings - Debug in Getting Started with Visual LANSA)

For example, assuming the computer name of the Development Environment that you want to use is mydevenv and the Visual LANSA Debug Service for that development environment is using network port number 51234, to debug the ENROL function in the PSLSYS process in the WEB partition, your URL might appear as follows:

#### http://www.lansa.com/cgi-bin/lansaweb? PROCFUN+PSLSYS+ENROL+WEB+BDEBUG+REMOTE=mydevenv:512

# 3.11 Invoking the iSeries Spool File Facilities

This feature is only available for the iSeries Data/Application Server. In order to use the Spool File Facility, it must be enabled using the LANSA for the Web Administrator or LANSA for the Web Servlet Administrator.

You can launch any of the Spool File facilities directly by entering the parameters in your call to the LANSAWEB CGI script program.

To retrieve the page for entry of spool file selection criteria, use the following parameter:

#### PRINT

For example:

#### http://www.lansa.com/cgi-bin/lansaweb?PRINT

To retrieve a list of spool files a PRINTLIST request is entered with the selection criteria as follows:

#### PRINTLIST+<user\_name>+<output\_queue>+<outq\_library>+ <form>+<userdata>+<status>

Note that each parameter is delimited by the plus (+) character.

<user\_name> is the name of the owner of the spool files which are to be shown in the list. This may be a specific user or \*ALL.

<output\_queue> is the name of the output queue which contains the spool files which are to be shown in the list. This may be the name of a specific output queue or \*ALL.

<outq\_library> is the library which contains the output queue previously specified. This may be a specific library name, \*LIBL or \*CURLIB.

<form> is the form type of the spool files which are to be shown in the list. This may be a specific form type, \*ALL or \*STD.

<userdata> is the user data of the spool files which are to be shown in the list. This may be a specific user data information or \*ALL. User data can contain lower case values. If you wish to enter a lower case value, enclose the parameter in single quotation marks.

<status> is the spool file status which is to be used for selection of spool files. This value may be one of the following:

*ALL	*CLOSED	*DEFERRED	*FINISHED
*HELD	*MESSAGE	*OPEN	*PENDING
*PRINTING	*READY	*SAVED	*SENDING
*WRITING			

For example:

http://www.LANSA.com/CGI-BIN/LANSAWEB? PRINTLIST+GROUPAUSR+qprint+\*LIBL+\*ALL+\*ALL+\*ALL or

#### http://www.lansa.com/CGI-BIN/LANSAWEB? PRINTLIST+GROUPAUSR+\*ALL+\*LIBL+\*ALL+'ListH5'+\*ALL

To display a spool file, a REPORT request plus details which identify the spool file are entered as follows:

#### REPORT+<job\_name>+<user\_name>+<job\_number>+ <spool\_filename>+<spool\_file\_number>+<from\_page>+<to\_page>

Note that each parameter is delimited by the plus (+) character. The details that identify the spool file are similar to those on the OS/400 command DSPSPLF.

<job\_name> is the name of the job which created the spool file.

<user\_name> is the name of the user profile under which the job which created the spool file was run.

<job\_number> is the system assigned job number of the job which created the spool file.

<spool\_filename> is the name of the spool file to be displayed.

<spool\_file\_number> is the number of the job's spooled file that is to be displayed.

<from\_page> is an optional parameter and may contain the number of the first page you wish to display. If this parameter is not provided it is assumed to be 1.

<to\_page> is an optional parameter and may contain the number of the last page you wish to display. If this parameter is not provided it is assumed to be the final page of the spool file.

For example: http://www.lansa.com/CGI-BIN/LANSAWEB? REPORT+WEB0000001+WEBUSER+'092926'+QPJOBLOG+'0001'+'1'+'2'

# 3.12 Event Logging

LANSA for the Web provides support for event logging. Event logging is also commonly known as 'click tracking'. Event logging allows you to track the navigation of the users of your Web Function Applications. It creates a log record for every page served up by LANSA for the Web. The information from the event logger can help you answer typical questions like:

- How long did the user stay at my site?
- How many pages did the user access at my site?
- How is the customer using my site?

LANSA for the Web also allows you to trace user defined data as well as system data. The data traced can be easily queried for data mining purposes.

You can trace two levels of information – summary information and detailed information. The summary information provides information about:

- Date and time the user first accessed your site. The page accessed is also logged.
- Date and time the last page was accessed.
- Number of pages accessed.
- Web user profile, if any.
- IP Address, if available.
- HTTP Referrer, if available.

The summary information is stored in a LANSA table, DC@W26.

The detailed information provides information about each page accessed, including:

- Date and time the page was accessed.
- User defined trace information.

The detailed information is stored in a LANSA table, DC@W27.

To remove recorded event logging entries in these files on the iSeries, call the Cleanup program as follows:

# CALL W3@P2210 PARM('<WEBPGMLIB>' '<DATE>') here

where

<WEBPGMLIB> is your lansa web program library

<DATE> is the date in YYYYMMDD format

# 3.13 Enabling Event Logging

If you want to enable event logging, you are required to include the <RDML

TRACE> tag in either your process specific layout page

(<process\_name>\_LAYOUT) or the default layout page

(DEFAULT\_LAYOUT). If you do not have a process specific layout page, it is suggested that you create your process specific layout page with a copy of the contents of the default layout page.

Enabling event logging using the layout pages permits you to have tracing over the entire application. By doing so, it is possible to tell how long a user has been using your application. Alternatively, you may enable logging over specific pages. This is done by putting <RDML TRACE> at the top of each your HTML pages.

The <**RDML TRACE**> tag is used to instruct LANSA for the Web to look up the trace page and determine which level of tracing is to be applied to your application. The appropriate tracing information will be written to the relevant files depending on the keywords specified in the trace file.

When determining the level of tracing, LANSA for the Web looks up either the process specific trace page (<proces\_ name>\_TRACE) or the default trace page, DEFAULT\_TRACE. This trace file must only contain a single line with either TRACE\_SUMMARY or TRACE\_DETAILED as the keyword. Note that the keyword is in upper case.

The TRACE\_SUMMARY keyword indicates that you only want to log summary information and the logging information is stored in the DC@W26 (summary) file. This is a default keyword in the DEFAULT\_TRACE page. A new trace identifier is assigned for each process or procfun request made.

The TRACE\_DETAILED keyword extends on the summary level tracing permitting you to log user defined fields. The detailed information is kept in the DC@W27 (detailed) file. This file contains information on how each of the pages was accessed for each trace identifier.

# **3.14 Logging User Defined Fields**

LANSA for the Web allows you to trace fields within your functions. If you want to trace information in your function, you will need to have event logging enabled at the detailed level and you must include a LANSA standard field, STDTRCFLD, in your RDML function. This is defined as a CHAR(100) field in the LANSA Repository.

The Event Logging facility allows you to trace up to 10 fields in your RDML function. The nominated field names are concatenated together into the STDTRCFLD field. The first trace field is defined in position 1 to 10 of the STDTRCFLD, the second trace field in position 11 to 20, and so on. For example, if you want to trace the EMPNO, SURNAME and GIVENAME fields, these field names are concatenated together. Each of the field names that are less than 10 characters are padded with blanks. In this case, the value of the STDTRCFLD field would be:

## EMPNO SURNAME GIVENAME

You use the STDTRCFLD field as a \*HIDDEN attribute as part of your display or request commands in your functions.

When detailed event logging is enabled in LANSA for the Web, it will check the value of the STDTRCFLD field to determine the user defined fields to trace. The field value length of each of the user defined trace fields is limited to 50 bytes. LANSA for the Web will truncate the data if the length is greater than 50. The trace fields and their contents are logged to the respective fields in the detailed file (DC@W27).

## 4. WEBEVENT Functions

WEBEVENT functions are designed to support the browser paradigm and the use of the Back button. To understand how to create WEBEVENT functions, refer to the following:

- 4.1 What is a WEBEVENT Function?
- 4.2 How Does WEBEVENT Work?
- 4.3 How is WEBEVENT Different?
- **4.4 WEBEVENT Templates**
- 4.5 WEBEVENT Example
- 4.6 Automatic Data Exchange
- 4.7 WEBEVENT Routing
- 4.8 WEBEVENT Keywords
- 4.9 Considerations for WEBEVENT Functions
- 4.10 WEBEVENT Data and Function Timeout

WEB003 - Coding WEBEVENT Functions



# 4.1 What is a WEBEVENT Function?

LANSA allows you to create a special type of RDML Web function specifically for use with the Internet. This Web function is called a WEBEVENT function.

The WEBEVENT function is designed specifically for the Internet user and the browser paradigm. In the browser paradigm, a persistent connection is not required. Each page is independent. You cannot dictate the navigation path of your application to the user. The Internet user has a Back button in the browser, which allows them to go back several pages and resubmit a request.

A function is defined to be a WEBEVENT function by specifying the following option in the RDML FUNCTION command:

#### **FUNCTION OPTIONS(\*WEBEVENT)**

With WEBEVENT functions, the LANSA function will terminate after a DISPLAY or REQUEST command is executed. No other logic will be executed after the HTML page for the function is sent. Using this technique, you must code your functions to properly process the information when the user decides to submit a screen with data. The logic of the RDML function must be written specifically for WEBEVENT. You cannot simply add the \*WEBEVENT keyword to your functions.

WEBEVENT functions cannot use any of the standard function keys, i.e. Exit, Cancel, Add, Change, Delete, Prompt, etc. because the function terminates once the display is sent. Instead, navigation can be controlled by keywords in the USER\_KEYS parameter of the DISPLAY or REQUEST command (or by using links from one page to another). Each description of the user key is a keyword that is used to link one LANSA function to another. Each of these user keys will be displayed as a button. When the user selects a button, the linked function will be called.

When the linked function is called, all form variables, hidden fields and any browse list data will automatically be exchanged with the called function. For more details, refer to 4.6 Automatic Data Exchange.

For a detailed example of writing WEBEVENT functions, refer to 4.5 WEBEVENT Example.



## 4.2 How Does WEBEVENT Work?

If you execute the standard FRENQ02 template to create a procedural header/detail style function, it generates a function with REQUEST and DISPLAY screens. The overall structure of the RDML in the function might appear as follows:

FUNCTION	OPTIONS(*DIRECT)
GROUP_BY	NAME(#HEADER)
DEF_LIST	LIST(#LIST)
BEGIN_LOOP	
REQUEST	FIELDS(#DEPTMENT)
FETCH	FIELDS(#HEADER)
SELECT	FIELDS(#LIST)
ADD_ENTRY	TO_LIST(#LIST)
ENDSELECT	
DISPLAY	FIELDS(#HEADER) BROWSELIST(#LIST)

```
••••
```

END\_LOOP

To convert this to a WEBEVENT function, you might divide the single function into two functions: FUNC001 and FUNC002. FUNC001 will REQUEST the information to be located and FUNC002 will DISPLAY the data.

The function structure would appear something like the following:

FUNC001:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
CHANGE FIELD(#DEPTMENT) TO(*DEFAULT)
REQUEST FIELDS(#DEPTMENT)...USER_KEYS((01 SEARCH))
```

FUNC002:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
GROUP_BYNAME(#HEADER)...
DEFINELIST(#LIST)...
FETCHFIELDS(#DEPTMENT)... NOT_FOUND(R01)
SELECTFIELDS(#LIST)...
ADD_ENTRYTO_LIST(#LIST)
ENDSELECT
```

R01: DISPLAYFIELDS(#HEADER)... BROWSELIST(#LIST)

When the user executes the LANSA WEBEVENT function, FUNC001 will simply send the REQUEST for the search data and then terminate. For example,

it might ask for a Department Code. Within FUNC001, a user key is nominated. This user key is called Search and is linked to FUNC002.

After the user enters the data, the Search button will be pressed. This button is a link to the FUNC002 function. FUNC002 is called and the input data from FUNC001 is passed from the browser to FUNC002. Because this is a WEBEVENT function, the variables from FUNC001 are passed to FUNC002 as if FUNC002 were being called from within FUNC001. This is automatically handled by LANSA.

FUNC002 will take the search parameters from FUNC001 and select the data from the files to build a browse list with the results. FUNC002 will display these results and then it will terminate. Like FUNC001, it can nominate user keys and linked functions. For example, it might link back to FUNC001 or it might link to a FUNC003 which provides details of a selected transaction.

If the user presses the Back button in the browser, they can return to the REQUEST screen in FUNC001 and enter new data. When the user presses the Search button, FUNC002 is simply called once again. The information is exchanged so that FUNC002 can execute properly.

**Note:** If you are using browse lists, the lists should match exactly in FUNC001 and FUNC002 so that the data is passed properly. For more details, refer to 4.6 Automatic Data Exchange.

For a detailed example of writing WEBEVENT functions, refer to 4.5 WEBEVENT Example.



WEB003 - Coding WEBEVENT Functions

# 4.3 How is WEBEVENT Different?

The WEBEVENT function is not coded like a procedural LANSA function. The key differences include:

- WEBEVENT functions automatically terminate immediately after sending the page to the user, that is, immediately after a REQUEST or DISPLAY statement.
- WEBEVENT functions are designed with their processing logic first. The last operation must be the REQUEST or DISPLAY. The logic of the RDML function must be written specifically for this style of function.
- WEBEVENT functions cannot use any of the standard function keys, i.e. Exit, Cancel, Add, Change, Delete, Prompt, etc. because the function terminates once the display executes.
- WEBEVENT functions control user navigation using the USER\_KEYS parameter of the DISPLAY or REQUEST command or by using links from one page to another. For details, refer to 4.7 WEBEVENT Routing.
- The LANSA function terminates after the REQUEST or DISPLAY so the transaction server is not required to maintain state. (**Note:** the transaction server is still used to process LANSA Web function jobs). The user is able to use the browser's Back button if they wish.
- The exchange of data between WEBEVENT functions is automatically handled by LANSA.
- WEBEVENT functions are not procedural. You must call your WEBEVENT functions directly using the PROCFUN keyword. These functions cannot be executed from a process menu.



# **4.4 WEBEVENT Templates**

LANSA includes web function application templates, which will create WEBEVENT functions. The templates include:

- FRWEBADD01 Data entry for \*WEBEVENT. FRWEBADD01 creates a simple data entry WEBEVENT function.
- FRWEBENQ01 Page at a time with detail display. FRWEBENQ01 creates a WEBEVENT function that has a generic search, and then displays the results of the search in a page at a time browse list. You can select entries from the browse list for a more detailed display of information.



# 4.5 WEBEVENT Example

In order to understand how to design and program WEBEVENT functions, the following step-by-step example has been included:

- 4.5.1 Procedural Add an Employee Function
- 4.5.2 Changes Required to Restructure for WEBEVENT
- 4.5.3 Handling Re-entrant Functions
- 4.5.4 Final WEBEVENT Function
- 4.5.5 More Complex Example

WEB003 - Coding WEBEVENT Functions

## 4.5.1 Procedural Add an Employee Function

Consider a very common data entry function such as the type of function created by the FRADD01 template.

FUNCTION	OPTIONS(*DIRECT)
GROUP_BY	NAME(#PANEL) FIELDS(#EMPNO #SURNAME)
MESSAGE	MSGID(DCU0010) MSGF(DC@M01) MSGDTA('employ
BEGIN_LOOP	
REQUEST	FIELDS(#PANEL) DESIGN(*DOWN) IDENTIFY(*DES
INSERT	FIELDS(#PANEL) TO_FILE(PSLMST)
MESSAGE	MSGID(DCU0011) MSGF(DC@M01) MSGDTA('employ
CHANGE	FIELD(#PANEL) TO(*NULL)
END_LOOP	

In this example, you have a single function, which must be converted to follow the rules of the WEBEVENT function. The behavior of the function should be as follows:

- When the function is first called, it simply displays a data entry panel and then the function ends.
- The user will enter data into the function and call the function again.
- When the data is passed to the function, it executes and attempts to insert the data to the file.
- If there are errors, then the error messages must be displayed.
- If there are no errors, a completion message is sent and the fields are reset.
- The function displays the data entry panel and the function ends.

Refer to 4.5.2 Changes Required to Restructure for WEBEVENT.

## **4.5.2 Changes Required to Restructure for WEBEVENT**

For this function to be re-written using the FUNCTION OPTIONS(\*WEBEVENT) style, the following changes must be made:

- REQUEST must be the last statement executed in the function, so the commands must be reorganized.
- The MENU\_KEY is not allowed in the REQUEST statement.
- BEGIN\_LOOP and END\_LOOP must be removed and replaced by statements to determine if the function is executing for the first time.

The overall changes required are shown below.

FUNCTION OPTIONS(\*DIRECT \*WEBEVENT)

GROUP\_BY NAME(#PANEL) FIELDS(#EMPNO #SURNAME...) If the function has data to be inserted...

INSERTFIELDS(#PANEL) TO\_FILE(PSLMST) VAL\_ERROR(...)MESSAGEMSGID(DCU0011) MSGF(DC@M01) MSGDTA('employeeCHANGEFIELD(#PANEL) TO(\*NULL)

End of the If logic.

REQUEST FIELDS(#PANEL) DESIGN(\*DOWN) IDENTIFY(\*DESC)

In order to determine if the function is executing for the first time or if it has data to be inserted, you need to use a variable to determine the state of the function. Refer to 4.5.3 Handling Re-entrant Functions.

## **4.5.3 Handling Re-entrant Functions**

In this example of a WEBEVENT function, a single function is being written to call itself. This example is described as re-entrant. The same function is reentered in order to process the Web page. This is a common approach as it allows the RDML logic associated with a page to be contained in the same function as the Web page itself.

In this function example, you must exchange a hidden field called RENTRY. This field tells the function if it is being called for the first time or if it is being called to process data.

For example:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
GROUP_BY NAME(#PANEL) FIELDS(#EMPNO #SURNAME... (REN]
DEFINE FIELD(#RENTRY) TYPE(*CHAR) LENGTH(1)
IF COND('#RENTRY *EQ Y')
INSERT FIELDS(#PANEL) TO_FILE(PSLMST)VAL_ERROR(T01)
MESSAGE MSGID(DCU0011) MSGF(DC@M01) MSGDTA('employee']
CHANGE FIELD(#PANEL) TO(*NULL)
ENDIF
```

T01 CHANGE FIELD(#RENTRY) TO(Y)

REQUEST FIELDS(#PANEL) DESIGN(\*DOWN) IDENTIFY(\*DESC)..

When the function is called for the first time, the RENTRY field will have a value of NULL. The function simply displays the REQUEST asking the user to enter data for the file. Notice that RENTRY is set to YES before the function terminates.

The user key for the SUBMIT button will indicate that the next function to be called is itself. (For details, refer to 4.8 WEBEVENT Keywords.)When the function is called a second time, the variables are passed to the function and the RENTRY field is YES. The function knows it must perform the insert to the data file.

Also note how the T01 label is used. If errors occur during the INSERT, the function must display the current data. The reset of the #PANEL fields to \*NULL is skipped.

Refer to 4.5.4 Final WEBEVENT Function.

## **4.5.4 Final WEBEVENT Function**

The add function can be re-written using the WEBEVENT style as shown below:

	FUNCTION	OPTIONS(*DIRECT *WEBEVENT)
	DEFINE	FIELD(#RENTRY) TYPE(*CHAR) LENGTH(1)
	GROUP_BY	NAME(#PANEL) FIELDS(#EMPNO #SURNAME #
	IF	COND('#RENTRY *EQ Y')
	INSERT	FIELDS(#PANEL) TO_FILE(PSLMST) VAL_ERRC
	CHANGE	FIELD(#PANEL) TO(*NULL)
	MESSAGE	MSGID(DCU0011) MSGF(DC@M01) MSGDTA ('e
	ENDIF	
T01	CHANGE	FIELD(#RENTRY) TO(Y)
	REQUEST	FIELDS(#PANEL) DESIGN(*DOWN) IDENTIFY(*

An entry must be added to the WEBEVENT Links table using the Web Function Editor to link this process/function (via the user key1) to itself. The function would present an HTML page of input fields, as well as a button. When the button is clicked, the function would call itself and pass the data to be inserted into the PSLMST file.

## 4.5.5 More Complex Example

Imagine you have a more complex RDML function, which has three screens. It might look something like the following example. Using the re-entrant technique, you use a CASE statement and direct the processing to the correct part of the function based on the last screen processed.

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
•••
CASE
       OF FIELD(#RENTRY)
        VALUE IS(= 1')
WHEN
CHANGEFIELD(#RENTRY) TO(2)
. . .
DISPLAY FIELDS(...) USER_KEYS((01 SUBMIT))
        VALUE_IS(= 2')
WHEN
         FIELD(#RENTRY) TO(*NULL)
CHANGE
DISPLAY FIELDS(...) USER_KEYS((01 RETURN))
OTHERWISE
CHANGE FIELD(#RENTRY) TO(1)
•••
REQUEST FIELDS(...) USER_KEYS((01 SEARCH))
ENDCASE
```

In this example, the #RENTRY field is set in each branch of the CASE statement in order to control how the function will execute the next time it is called.

## 4.6 Automatic Data Exchange

LANSA automatically handles all data interchange between WEBEVENT functions.

When a linked function is called from your WEBEVENT function, the following will automatically be exchanged:

- all form variables
- hidden fields
- all browse list data.

It is very important that the browse list definitions must be identical in all respects in all called functions as well as the current function being executed. If you have more than one DEF\_LIST command in your function, all the browse lists will be exchanged even though you are only displaying one of the browse lists in your function.

If you exceed the LANSA exchange list limit, you can turn on the *Enable Extended Exchange* flag in the Web Administrator. Once you have enabled this option, you will have to recompile your functions. The *Enable Extended Exchange* flag is on the Miscellaneous tab of the *Configure Data/Application Server* in the LANSA for the Web Administrator. LANSA for the Web Administrator will have been installed when you installed LANSA for the Web. Refer to the *Exchange List* item in the Summary of Platform Differences.

When you create WEBEVENT functions, you MUST NOT use commitment controls as each function is effectively a new job. This style of application does not cater for database cursors.

To understand how and when this data is purged, refer to 4.10 WEBEVENT Data and Function Timeout.

# **4.7 WEBEVENT Routing**

There are several techniques which can be used to link your WEBEVENT functions:

### 1. USER\_KEYS and KEYWORDS

Navigation from one WEBEVENT function to another can be controlled by keywords in the USER\_KEYS parameter of the DISPLAY/REQUEST command. Each description of the user key is a keyword that is used to link one LANSA function to another. Each of these user keys will be displayed as a button. When the user selects a button, the linked function will be called. The keywords and their linked functions are set up using the Web Function Editor. For more details, refer to 4.8 WEBEVENT Keywords.

### 2. Web Link Web Components

Web link components are only used with WEBEVENT functions. You can use Web link components to link to other functions, instead of using Keywords.

Web link components also allow you to display the links as images instead of buttons. These components allow you to dynamically change the links as well as the presentation of the links, without having to recompile your WEBEVENT functions. For more details, refer to Web Link.

### 3. Standard HTML Links

If you wish to link to another function but do not want to pass any information to the next function, you may use a standard hypertext link.

### 4. JavaScript

Using the HandleEvent function in the default JavaScript used by LANSA Web functions, you can control the navigation between WEBEVENT functions.

## 4.8 WEBEVENT Keywords

To control the application flow in WEBEVENT functions, you can use the USER\_KEYS parameter in your RDML function along with Keywords in the Web Function Editor.

For example:

```
REQUEST FIELDS(#PANEL) DESIGN(*DOWN) IDENTIFY(*DESC) E
```

The USER\_KEYS parameter will cause a button to appear on the function.

The keywords and their linked functions are set up using the Web Function Editor. Select the Tools action bar category and the Keywords option. You can add new links or maintain existing links. For more details, refer to Keywords.

	Language: Description:
Process:	ENG 🔽
function:	Linked Process:
(eyword:	Linked Function:

The Process, Function and Keyword parameters are used to identify the specific process and function and user key being pressed. The Linked Process and Linked Function are the names of the process and function to be called when the button is pressed. In the case of a re-entrant function, the names will still remain the same. The Description is used to define the words which will appear on the button.

**Very Important:** The keyword is case sensitive! If you use "SUBMIT" in uppercase in your RDML USER\_KEY statement, then you must use "SUBMIT" and not "submit" when defining the keywords.



## **4.9 Considerations for WEBEVENT Functions**

Following are some important considerations when creating WEBEVENT functions:

- You cannot call WEBEVENT functions from a process menu. You must not use the PROCESS keyword in your URL syntax. Since WEBEVENT functions terminate as soon as a DISPLAY or REQUEST command is executed, the LANSA job is terminated.
- You must call WEBEVENT functions directly. You must use the PROCFUN keyword in your URL syntax.
- Navigation between WEBEVENT functions can be handled by User Keys and Keywords.
- If you are creating the function links manually, you should use the HandleEvent JavaScript function to process the call instead of using an anchor block (<A HREF>) tag. The HandleEvent function redirects the request to the same Web job at the Data/Application Server whereas the anchor block request starts a new Web job.
- If you use navigation by using an HTML <A HREF=URL>, no data will be passed to the called WEBEVENT function.
- WEBEVENT functions must not use commitment control as each function is effectively a new job. This style of application does not cater for database cursors.
- WEBEVENT functions cannot use function keys such as Exit, Cancel, Add, Change, Delete, Prompt, etc. because the function has already terminated.
- You must check the STDHEADER page you are using for your WEBEVENT function. All function keys must be enclosed within an <RDML BUTTON> LANSA tag. If a function key does not have an RDML BUTTON tag, you will need to enclose these function keys with a pairing of <RDML BUTTON="&WEBEVENT"> and </RDML> tags. Your header should have no function keys, i.e. no buttons.
- WEBEVENT functions expect to have only one REQUEST or DISPLAY screen used per function. This does not mean that you cannot have more than one REQUEST statement, but simply that in the flow of logic only one statement should be encountered. A compiler warning message may be displayed if more than one REQUEST or DISPLAY is used in the function.
- There will be RDML executed after the REQUEST or DISPLAY. The

WEBEVENT function will terminate after the DISPLAY/REQUEST command is executed. No other logic will be executed after the HTML page for the function is sent. A compiler warning is issued if statements are encountered after the REQUEST or DISPLAY.

- Do not use DISPLAY or REQUEST or POP\_UP commands in a subroutine. These commands are not allowed in subroutines for \*WEBEVENT functions.
- Output fields are not exchanged between WEBEVENT functions. When the linked function is called, all form variables, hidden fields and any browse list data will be exchanged to the called function.
- If you are using browse lists, the lists should match exactly in the calling and called WEBEVENT functions so that the data is passed properly.
- Check that your INSERT or UPDATE or other validations commands do not use \*LASTDIS.



# 4.10 WEBEVENT Data and Function Timeout

It is important to understand how WEBEVENT function data is handled as it relates to the jobs being maintained by the transaction server.

A LANSA WEBEVENT function terminates as soon as it has processed a DISPLAY or REQUEST command. However, the Web job is reserved for the user, pending further interaction unless you have coded the function to terminate the job immediately.

The LANSA for the Web Transaction Monitor will terminate a WEBEVENT function if the timeout period has elapsed. However, if you continue to interact with this WEBEVENT function, LANSA will automatically allocate a new job for the request and handle the data interchange. Your data is still intact. (You must not use the Job Identifier as the unique identifier, as any restarted WEBEVENT functions may not be allocated to the same job at the Data/Application Server.)

When WEBEVENT functions time out, LANSA maintains the data for these jobs temporarily at the Data/Application Server. When the user interacts with these functions subsequently, the temporary data is restored and the user is not impacted – a new LANSA job has been allocated and the temporary data restored for the function.

LANSA for the Web holds the temporary data for a specified period of time. This temporary data is cleared when the inactive time of the job exceeds this purge time period. The purge period time is set in the *Purge WEBEVENT function data* option on the Transaction Monitor tab when you configure the Data/Application Server using the LANSA for the Web Administrator. If the user interacts with the function after the purge time, browse list data will not be exchanged as it has been purged. For more details, refer to the *Installing LANSA on Windows* Guide.

## **Terminating WEBEVENT Jobs Immediately**

LANSA Web functions provide you with a LANSA tag, <RDML MERGE="&END">, to indicate that the Web job is to be freed immediately (after the DISPLAY or REQUEST command) and returned to the pool of prestarted Web jobs.

This tag should only be used for functions, which have no user interaction once the function terminates. It should not be used for functions that have a browse list, which must be exchanged with a subsequent function. For more details, refer to Using <RDML MERGE="&END">.

# 5. LANSA Generated HTML Pages

LANSA Web Functions generate pages meeting both the HTML 4.0 standard and the XHTML 1.0 standard. In this guide, the term HTML will be used to describe generated HTML/XHTML pages. For details about XHTML, refer to LANSA for the Web XHTML.

LANSA Web Functions will generate the required HTML pages when you compile your functions. If you are planning on modifying these pages, you should review the following:

- 5.1 Page Security
- 5.2 Identifying Generated Pages
- 5.3 Modifying the Process Menu
- 5.4 Versioning of Pages
- 5.5 Comparing Versions
- 5.6 HTML Page Structure
- 5.7 LANSA Field Names in HTML Pages
- 5.8 HTML Generation Skeleton



# 5.1 Page Security

Under a traditional HTML development, each HTML page is an HTML document that resides in a directory. Your application may require hundreds of HTML documents scattered over numerous directories. The HTML pages are exposed to anyone who can gain access to the directories. The HTML pages can also be altered without your knowledge.

Using LANSA Web functions, the HTML pages are kept as LANSA internal files for easier backup/restore and maintenance procedures. The HTML pages cannot be accessed without using the Web Function Editor which prohibits unauthorized editing.

LANSA Web Functions also include special tags which are stored in HTML pages. These tags are processed when the pages are read from the LANSA internal files and then served to the client browser or computing device.

The images files, static pages and embedded objects (audio, video, etc.) which are used as part of the pages must be stored in the Web Server directories.

## **5.2 Identifying Generated Pages**

In order to modify the LANSA generated HTML pages, you will use the Web Function Editor. You must also know how to identify the pages associated with processes and functions.

When LANSA processes are compiled, no HTML is generated for the process. When a Web enabled LANSA process is executed, LANSA automatically generates the process menu page based on the DEFAULT\_INDEX and DEFAULT\_CONTENT pages. The DEFAULT\_INDEX page is used to initialize the Menu Area of the browser. The DEFAULT\_CONTENT page is used to initialize the contents area of the process menu. For more details, refer to 5.3 Modifying the Process Menu.

When a LANSA Web function is compiled, LANSA generates an HTML page for each REQUEST or DISPLAY statement within the function. Each HTML page adopts the following naming convention:

certain content /process name> function name> display sequence>

The <process name> is the name of the LANSA process. If the length of the process name is less than 10 characters, it is padded with spaces.

The <function name> is the name of the LANSA function. If the length of the function name is less than 7 characters, it is padded with spaces.

The <display sequence> is the logical sequence of the screen in the function. For example, the first screen displayed, a REQUEST statement, in the FUNC01 function of PROC01 process will adopt the name:

PROC01 FUNC01 001

and the second screen displayed, a DISPLAY statement, will adopt the name:

PROC01 FUNC01 002

Refer to Open . . .(XML/HTML mode).

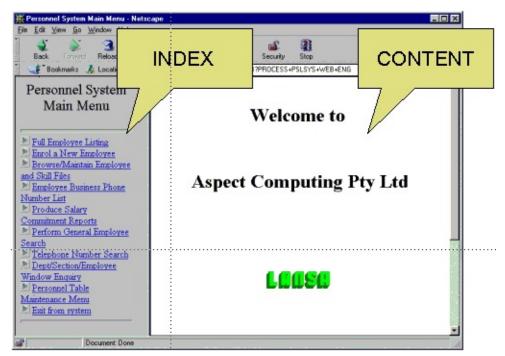
WEB004 - LANSA Generated HTML Pages

# 5.3 Modifying the Process Menu

LANSA does not create a stored HTML page for a process. The process menu HTML page is automatically generated based on the current process definition and the default HTML process page definitions. The LANSA Process Menu is used for your procedural Web applications. (If you are creating WEBEVENT functions, you should call them directly using the PROCFUN keyword.)

The process menu is divided into two areas:

- process index (default\_index) and
- process content (DEFAULT\_CONTENT).



The process index lists the functions which can be executed in the process. The list of functions is built when the process is called. The rules are based on the LANSA process definition. For example, the function control table will determine the order in which the functions are listed.

The generic HTML for the process index is stored in DEFAULT\_INDEX. If the HTML in this file is changed, it will impact all of the LANSA processes in the partition. You should create a process specific version of this page. Refer to Customizing Process Specific Pages.

The process content is an information area of the process menu and the HTML for the process content is stored in DEFAULT\_CONTENT. If the HTML in this

file is changed, it will impact all of the LANSA processes in the partition. You should create a process specific version of this page. Refer to Customizing Process Specific Pages.

# **5.4 Versioning of Pages**

When a Web enabled LANSA function is compiled, the HTML pages for the function are stored in a LANSA internal file. The pages in this file can be edited using the Windows-based Web Function Editor. (Since the pages are stored on the Data/Application Server, the Editor executes as a client/server application.) When you manually edit the HTML, you now have a version of the HTML which does not match the original version that was generated by LANSA.

LANSA allows you to save the previous versions of your pages. The LANSA for the Web Administrator allows you to define whether or not you want to save the previous versions of the generated page. It also allows you to specify the number of sets you want to save. Up to 10 versions can be saved. When you open an HTML page, the Web Function Editor allows you to specify the version of page to be accessed.

Each time a Web function is (re)compiled and HTML generation is selected, LANSA generates a new version of the pages for each screen. Any manual changes you have made can then be copied from the previous version back into the current version. For example, when a Web function is first compiled, the page will be version 0. Now you edit this page. It is still version 0. When the Web function is recompiled, version 0 would become version 1 and the newly compiled version becomes version 0. The most recent or current version is always version 0. The higher the number the older the version. So version 0 =current version, version 1 = previous version, version 2 = (previous - 1) version, etc.

When LANSA generates the HTML for a particular Web function, it does not check to see if the current page (if any) has been modified. The backup feature must be enabled to prevent LANSA from overwriting the manually edited version of the page when the recompile is performed. If the backup feature is enabled, LANSA will save the current page before generating a new page for the Web function being compiled.

**Note:** You can only save changes to version 0. You may open any version of a page, but you can only save the page as version 0. For example, if you open version 3 of the page and make changes, it cannot be resaved as version 3. It can only be saved as version 0. This rule ensures that your previous versions are not corrupted.

For more details, refer to the Web Function Editor Open (Basic) mode and Save As features.



## 5.5 Comparing Versions

The Web Function Editor provides a special compare and contrast feature for comparing versions of your pages. You can open an HTML page and then select another version of the page to compare with the opened page. The Editor will highlight the differences in the pages as you scroll through the documents.

You can configure the colors used to identify the inserted and deleted records in the documents. The HTML which was inserted is shown in one color, followed by the HTML, (if any), which was deleted shown in another color. For example, when a new line of HTML is simply inserted, it will be shown as an inserted line. When a line of HTML is changed, it will show the new inserted line and the old original line of HTML (deleted) which was changed.

The Editor supports both horizontal and vertical split screens when comparing pages .You can specify synchronized scrolling so that the two documents scroll at the same time.

The compare and contrast features are useful in cases where you have modified the page of a function and then recompiled the function. Using compare and contrast, you can quickly identify your modifications and reapply them to the latest version of the function. You could either open the most recent version and compare the changes from the earlier version, or you could open the earlier version and compare the changes from the most recent generation.

For more details, refer to Web Function Editor Compare feature.

### 5.6 HTML Page Structure

When you compile a LANSA Web function, an HTML page is created for each display (REQUEST or DISPLAY) in the function. Each display is divided into a number of page components:

- function layout (including default JavaScript and styles)
- standard header
- function HTML
- standard footer.

The page components are embedded using special LANSA tags such as <RDML LAYOUT>, <RDML MERGE> and <RDML COMPONENT>. For more details, refer to RDML Tags.

For example, a typical LANSA Web function might produce the following generated HTML page:

<rdml layout=""> <!-- Process : xxxxxxxx Test Process<br--><!-- Function : xxxxxx Display Sections<br--><!-- Page : 001</th--><th>&gt;</th><th>&gt; &gt;</th></rdml>	>	> >
Generated by - xxxxxxxx<br Created by user - xxxxxxxxx<br Time and Date - xxxxxxxxxxx</td <td></td> <td>&gt; &gt; &gt;</td>		> > >

<form action="/<RDML MERGE="&CGI">/LANSAWEB?FUNCTION+ <RDML MERGE="&SESSION">" method="post" name="LANSA">

<RDML MERGE="&HIDDEN">

### <RDML COMPONENT="STDHEADER">

```
<center><h1><RDML MERGE="@FUNCTION"></h1></center>
```

...main body of function will appear here...

#### <RDML COMPONENT="STDFOOTER">

#### </form>

When the page is presented to the client browser, there will be just one HTML page. For the developer, each of the page components can be separate documents which can be shared and reused. For more details about these components, refer to Standard HTML Page Components.

Using standard page components for items like layout pages, headers, footers, etc. allows the developer to create common and shared definitions for the HTML pages. This approach greatly increases productivity because the developer does not need to repeat common information. Application maintenance is also significantly reduced as the developer need only change a single definition instead of modifying each function.

LANSA ships a set of standard HTML page components. These standard page components can be embedded into other HTML pages or they can be used as default pages. Standard page components are used to minimize the maintenance effort. For more details about these components, refer to Shipped Default Pages.

For example, the component STDHEADER displays a company logo and includes all the application navigation buttons. The STDHEADER is embedded into the HTML generated for every LANSA Web function. To change the company logo, you would only need to modify one file, STDHEADER. You do not need to modify the HTML in every function.

Each LANSA Web function will include a standard header and a standard footer in the generated HTML. LANSA Web functions allow you to use various styles for your standard header. You can choose between a static

(STDHEADER\_STYLE1) or a dynamic header. If you choose to have a dynamic header, you have the additional choice of implementing either a scrolling (STDHEADER\_STYLE2) or a non-scrolling

(STDHEADER\_STYLE3) header. LANSA allows you to configure a header which can be used for all pages for a particular LANSA process. For more details, refer to Header Styles.

## 5.7 LANSA Field Names in HTML Pages

When you review the HTML generated for your LANSA Web functions, you may notice that your LANSA field names have a one character prefix added to each field name. The LANSA generated HTML uses the following specific field naming conventions:

A<field name> = Alphanumeric fields

P<field name> = Packed numeric fields

S<field name> = Signed numeric fields

L<field name> = Lowercase alpha fields.

For example, if your RDML contains a field #NAME, where #NAME is an alphanumeric field, then you will see a field name ANAME used in your HTML page. If field #COUNT is packed, then you will see PCOUNT.

If you customize your HTML page, you must use this naming convention.

This field naming convention allows LANSA to overcome some of the limitations of HTML and the browser where there is no designation of field types. The HTML page and the browser do not know the difference in the type of field. This naming convention is only used by LANSA.

## **5.8 HTML Generation Skeleton**

LANSA allows you to control the generated HTML page structure or skeleton. If you are planning to modify the HTML generation skeleton, you should review the following:

5.8.1 What is the HTML Skeleton?

5.8.2 How Do I Use the HTML Skeleton?

5.8.3 Considerations for Using the HTML Skeleton

# 5.8.1 What is the HTML Skeleton?

LANSA provides an HTML skeleton or template to generate the HTML pages for your LANSA Web functions.

LANSA allows you to customize the HTML skeleton. The default HTML skeleton WEBSKEL is contained in the LANSA data library. (For LANSA for iSeries, file DC@F28 is used. For Visual LANSA, a WEBSKEL.S file is used.) This skeleton must not be modified.

If you wish to create your own customized HTML skeleton, you should create the skeleton in a new source member. (For LANSA for iSeries, create a member in the DC@W22 file. For Visual LANSA, create a <filename>.S file in the partition source directory.) You can create a default skeleton or you can create a skeleton which is specific to a partition, process or function.

When the HTML for a Web function is generated, LANSA will search for source members to be used as the HTML skeleton. The following search order is used:

- function name for a WEBEVENT function only
- process name
- partition name
- special value 'DEFAULT'.

If none of these source members are found, then the default shipped HTML skeleton WEBSKEL will be used.

Sample HTML skeletons are shipped. For example, file member named SSISAMPLE is an example of an SSI HTML skeleton and looks like this:

%W3FLD

```
<br />
```

%W3BRW

<RDML MERGE="&END">

# 5.8.2 How Do I Use the HTML Skeleton?

If you identify a standard change which needs to be made to the structure of all of your Web functions in a process or in a partition, then you have identified an opportunity to use an HTML skeleton to make this structural change. Creating your own HTML skeleton allows you to control some parts of the default structure of the HTML generated by LANSA Web functions. For example, if you do not want to use a STDFOOTER in your Web functions, it can be removed from the skeleton.

For example, the current HTML skeleton uses the same background for all of your LANSA Web functions. The graphic variable \*LW3CLNTBKGND is used to control the client background. If you want to create a specific background for your applications in a partition or in a process or in a specific function without manually editing the HTML each time, you can create a new skeleton.

To create a client background which is partition specific, you need to do the following:

- 1. Create a new text Web component called MYBACKGND which will be used to store the specification for the client background you wish to use. Remember that the Web component is stored at the partition level.
- 2. Edit the HTML skeleton and replace the line:

```
<body bgcolor="<RDML MERGE="*LW3CLNTCOLOR">"<br/>background="
```

```
<RDML MERGE="*LW3CLNTBKGND">" <RDML COMPONENT="FOR! with:
```

```
<body bgcolor="<RDML MERGE="*LW3CLNTCOLOR">"<br/>background="
```

```
<RDML COMPONENT="MYBACKGND">" <RDML COMPONENT="FOI
```

3. Save the customized HTML skeleton using the Partition Name.

When your Web function HTML is generated in the specific partition, it will now use your new Web component to define the client background.

Once changes are made to the skeleton, you must recompile your Web functions and regenerate the HTML for the changes to take effect.

### **5.8.3 Considerations for Using the HTML Skeleton**

Following are some important considerations when customizing the Web skeleton:

- Never modify the shipped HTML skeleton WEBSKEL (stored in DC@F28 for the iSeries, or WEBSKEL.S for Windows) as your modifications will be overwritten when a new version is shipped.
- Save your customized skeletons in DC@W22 for the iSeries or in the source partition directory.
- As new features are added to LANSA Web functions, the default Web skeleton may change from one version to the next.
- You can use the IBM Source Edit Utility (SEU Editor) or the command Edit File (EDTF) to modify the HTML skeletons if you are using an iSeries Data/Application Server. For a Windows Server, you can use a general text editor.
- Never modify any of the special tags beginning with %, for example %W3FRM or %W3BRW.
- Be very careful with changes to the table format.
- Be sure to test your changes to the skeleton in a variety of situations.
- Make sure that all developers know when you have customized the skeleton if changes are made at the partition level.
- You can use Web components and graphic variables in your skeleton.
- Function level Web skeletons can only be created for WEBEVENT functions.
- Once changes are made to the skeleton, you must recompile your Web functions for the changes to take effect.
- If you are exporting your application, remember to export your Web skeleton if you are planning on recompiling your application.

# 6. Default Process Pages

LANSA Web functions use a number of default HTML pages. These pages can be used to enhance your Web function applications. To understand how to use these pages, review the following:

- 6.1 What are Default Process Pages?
- 6.2 Shipped Default Pages
- 6.3 Other Default Pages
- 6.4 Customizing Process Specific Pages
- 6.5 User Defined Default Pages
- 6.6 Standard HTML Page Components
- 6.9 Process Specific Page Components



WEB005 - LANSA Process Pages

## 6.1 What are Default Process Pages?

LANSA Web functions use a number of default HTML pages to define the default structure of the generated HTML pages. The default pages are prefixed with "DEFAULT\_".

For example, the JavaScripts used in LANSA Web functions are stored in a page named DEFAULT\_SCRIPT. LANSA processes use two default HTML pages, DEFAULT\_INDEX and DEFAULT\_CONTENT to create the process menu and area content.

Whenever LANSA encounters a request to use a default page, it will search for a process specific version of the page. If a process specific page is not found, the default is used. This feature makes LANSA Web functions very flexible. For example, you can define the default presentation of your process menus by editing DEFAULT\_INDEX and DEFAULT\_CONTENT. If you create a process specific version of the INDEX and/or CONTENT page, LANSA will substitute these automatically without any HTML editing or RDML recompiling by the developer. For more details, refer to 6.4 Customizing Process Specific Pages.

The default pages are partition specific. You can create your own user defined default pages. Refer to 6.2 Shipped Default Pages and the installation guide for your platform for more details about the shipped default pages.

# 6.2 Shipped Default Pages

Following is a list of the default pages shipped with LANSA Web functions. You should avoid changing the contents of these pages as LANSA may update them in future releases. For details of how to create a process-specific version of these pages, refer to 6.4 Customizing Process Specific Pages.

These pages are partition-specific. They must be loaded as part of the partition set up.

If you modify these pages, you must remember to export them if you move your application to another partition or system. In order to select these pages for export, you must register them as Web components. Refer to for information about Web Application Deployment.

Page Name	Description
DEFAULT_CONTENT	The Default Content page is displayed in the Content area of the browser when a process is called.
DEFAULT_FRAMESET	Contains the default non-scrolling header style frameset. One frameset holds the STDHEADER page while the other frame holds the body of your function.
DEFAULT_HIDDEN	Contains the hidden fields used to exchange information. If you want to include other hidden fields for your Web function application, you can append the hidden fields to this page.
DEFAULT_INDEX	Default Process Menu page. This is the default process menu-handling page.

DEFAULT_LAYOUT	Contains the layout used for any LANSA processes that do not have a customized layout. The default layout assumes a character mode look-and-feel.
DEFAULT_SCRIPT	<ul> <li>Contains all the JavaScript functions shipped with the product. The page shipped with the product has certain sections conditionally disabled. These sections are for:</li> <li>static header support (no longer recommended)</li> <li>the calendar control support</li> <li>numeric checking</li> </ul>
	<ul> <li>DBCS support.</li> <li>If you need to enable any of these sections, you need to edit this page and remove the appropriate <rdml oncondition=""> tag.</rdml></li> <li>Refer to tutorial WEB005 - LANSA Process Pages.</li> </ul>
DEFAULT_STYLE	Customize this page to embed cascading style sheets for any LANSA process. Refer to Cascading Style Sheets.
DEFAULT_TRACE	<ul> <li>When using event logging, this page is used to decide on the level of tracing for your Web function application.</li> <li>By default, the summary level of tracing is used and requires this page to contain only the keyword TRACE_SUMMARY.</li> <li>A more detailed level of tracing requires this page to contain the keyword TRACE_DETAILED. This level of tracing is an extension of the summary tracing by providing information about how each page is used.</li> </ul>

For more default pages, refer to 6.3 Other Default Pages.

From the LANSA for the Web Administrator, Enable Partition copies the shipped default pages according to your partition's default language. If the shipped default pages do not exist for your partition's default language, then the English pages will be used.

For translation purposes, it is possible to copy default pages by language other than that of your partition's default language by using the W3@P2500 program. The parameters would be:

Parameter	Туре	Description
Partition	CHAR(3)	Partition to be enabled.
Partition Language	CHAR(4)	One of the partition's defined languages. It is not necessarily the default language.
Shipped Language	CHAR(4)	The shipped language default pages to copy from.
For example: CALL W	/3@P2500 ]	PARM('DEM' 'FRA' 'ENG')

This command copies the shipped ENG pages to FRA in the DEM partition.

### **6.3 Other Default Pages**

LANSA Web functions also allow you to create other default pages. These are the pages that are not shipped with the product. You can create these pages manually (using the Web Function Editor) to define the default attributes of your Web function application.

Page Name	Description
DEFAULT_HEADER	This contains header information that is typically encap by the <head> and </head> tags.
	If you want to use a specific header for a particular LA Web function, refer to the < <u>RDML LAYOUT</u> > LANS.
DEFAULT_HMENU	This contains the menu items for the Horizontal Menu component. Typically, you should use the e-Business Framework Wizard to create this page.
DEFAULT_LMENU	This contains the menu items for the Left Menu compo Typically, you should use the e-Business Framework V create this page.
DEFAULT_MSGPRES	Default presentation of LANSA messages in your Web function application. By default, the LANSA messages presented in a list box.
	You can override the format of the message presentatic using this page and defining your own message presen layout. If this page exists, it is used to present the LAN messages in your Web function application.
	An example of a DEFAULT_MSGPRES is shown below replaces the list box format with a list of messages.
	<img *lw3imgmessages"="" src="&lt;br&gt;&lt;RDML MERGE="/> " borc

	< <li><li><li><li><li><li><li><li><li><li< th=""></li<></li></li></li></li></li></li></li></li></li>
DEFAULT_RMENU	This contains the menu items for the Right Menu comp Typically, you should use the e-Business Framework V create this page.
DEFAULT_STYLE	Default customized styles for your Web function applic This page contains the cascading style sheets (CSS) de for your applications. By default, your application adopts the presentation sty configured for the browser. If you want to override any styles, you can achieve this by creating this default pag An example of DEFAULT_STYLE is shown below. It overrides the styles used for the  and <h1> tags <style> table {font- family: <RDML COMPONENT="FONTPREF"> font-size: 10pt; font-weight: normal}</td></tr></tbody></table></style></h1>

h1 {fontfamily: <RDML COMPONENT="HEADER1FONT font-size: 12pt; font-weight: bold} </style> For more details, refer to Cascading Style Sheets.

For more default pages, refer to 6.2 Shipped Default Pages.

If you modify these pages, you must remember to export them if you move your application to another partition or system. In order to select these pages for export, you must register these pages as Web components. Refer to for information about Web Application Deployment.

## 6.4 Customizing Process Specific Pages

LANSA Web functions allow you to customize any of the default pages to be process specific. Whenever LANSA encounters a request to use a default page, it will search for a process specific version of the page. If the process specific page is found, it will automatically be used. If it cannot find a process specific page, it will then use the default page.

To create a process specific page, use the following naming convention:

```
<process name>_<page identifier>
```

where <process name> is the name of the LANSA process and <page identifier> is the name of the 'default' page you want to customize for the process.

For example, if you want to customize a process specific Script page for the PSLSYS process, you would create a page named PSLSYS\_SCRIPT.

From time to time, LANSA may amend the default pages, either during an upgrade or via an EPC (Expedited Program Change). If such a change occurs, you must ensure that this change is reflected in all process specific pages that you have created.

Examples of shipped DEFAULT pages which can be customized:

DEFAULT_CONTENT	Default Process Content
DEFAULT_FRAMESET	Default Frameset
DEFAULT_HIDDEN	Default Hidden fields
DEFAULT_INDEX	Default Process Menu page
DEFAULT_LAYOUT	Default Layout
DEFAULT_SCRIPT	Default Script Page

Example:

If a process is called PAYROLL, the process specific default pages would be created as:

PAYROLL_CONTENT	Default Process Content
PAYROLL_FRAMESET	Default Frameset
PAYROLL_HIDDEN	Default Hidden fields
PAYROLL_INDEX	Default Process Menu page
PAYROLL_LAYOUT	Default Layout
PAYROLL_SCRIPT	Default Script Page

If you want to append additional lines to the default page for a specific process,

you can do this by using the <RDML PAGE> tag. For more details, refer to <<u>RDML PAGE></u>.

**Caution:** When exporting a web enabled function, process pages with the above naming convention for the associated process are also exported. For example, a function in process PAYROLL when exported will export all HTML pages for the function and all HTML pages starting with the characters 'PAYROLL\_'. Therefore, ensure that you do not create HTML pages that follow this naming convention that conflict with the name of a process.

All process specific pages are automatically exported with the LANSA process definition. These pages do not have to be registered as Web components.



WEB005 - LANSA Process Pages

### **6.5 User Defined Default Pages**

LANSA Web functions provides you with facilities to exploit the functionality of the default pages. This allows you to create your own user defined default pages that may have process specific versions for certain LANSA processes.

To use this facility, the syntax of the default page is:

DEFAULT\_UD<page name>

where <page name> is the name you have assigned to the page. The process specific page will then adopt the naming convention of:

```
<process name>_UD<page name>
```

where <process name> is the name of the LANSA process.

For example, if you are using cookies in your LANSA Web function applications, you might create a page named:

DEFAULT\_UDCOOKIES

To define cookies for the PSLSYS process, you would create a page named:

PSLSYS\_UDCOOKIES

You can access this user defined page by using the <<u>RDML MERGE</u>> tag with the special <u>Reserved Words</u> as follows:

<RDML COOKIES="&UDCOOKIES">

For more details of this example, refer to <<u>RDML COOKIES</u>>.

## 6.6 Standard HTML Page Components

These standard HTML page Web components are used by the shipped LANSA Web templates.

Page	Description
STDFOOTER	Standard Footer contains a standard set of buttons for LANSA.
	By default, this is included as a LANSA Web component in every LANSA Web function generated.
STDHEADER	Standard Header contains a display of the company logo and a standard set of buttons for LANSA.
	By default, this is included as a LANSA Web component in every LANSA Web function generated.
STDMORE	Web component used by the shipped Web templates.
STDNEXT	Web component used by the shipped Web templates.
STDPREV	Web component used by the shipped Web templates.
STDPROCFOOTER	Standard Footer used for LANSA processes.
STDSEARCH	Web component used by the shipped Web templates.
STDSELECT	Web component used by the shipped Web templates.

## 6.7 Other HTML Page Components

These standard HTML page Web components are also shipped with LANSA Web functions:

Page Description

STDHEADER\_STYLE1 STDHEADER page for Style #1 header style. This is the static header style. If you want to use this header style, copy this page to be the STDHEADER page in your system.

> For a description of the various header styles supported by LANSA Web functions, refer to Header Styles.

STDHEADER\_STYLE2 STDHEADER page for Style #2 header style. This is the dynamic header style. If you want to use this header style, copy this page to be the STDHEADER page in your system.

> For a description of the various header styles supported by LANSA Web functions, refer to Header Styles.

STDHEADER\_STYLE3 STDHEADER page for Style #3 header style. This is the non-scrolling header style. If you want to use this header style, copy this page to be the STDHEADER page in your system.

> For a description of the various header styles supported by LANSA Web functions, refer to Header Styles.

STDPRINT Default page for entry of spooled file selection criteria.

This is the default page that is displayed in response to a PRINT request to the CGI script program LANSAWEB. LANSA will use a MY\_STDPRINT page if available, otherwise it will use the STDPRINT page. These pages are retrieved from partition "WEB", language "ENG".

	If you do not have a partition "WEB", contact your local LANSA product vendor for technical support.
STDREPORTLIST	Default page for display of spooled file list
	This is the default page that displays the list of spooled files that match the selection criteria entered in the STDPRINT page. LANSA will use a MY_STDREPORTLIST page if available, otherwise it will use the STDREPORTLIST page. These pages are retrieved from partition "WEB", language "ENG".
	If you do not have a partition "WEB", contact your local LANSA product vendor for technical support.
STDREPORT	Default page for display of a spooled file details.
	The details of the requested spooled file are merged with this page and displayed in response to a REPORT request to the CGI script program LANSAWEB. LANSA Web will use MY_STDREPORT page if available, otherwise it will use the STDREPORT page. These pages are retrieved from partition "WEB", language "ENG".
	If you do not have a partition "WEB", contact your local LANSA product vendor for technical support.

### 6.8 Special XHTML Pages

These standard XHTML pages are shipped with LANSA Web functions. They are used to embed the DOCTYPE declaration in XHTML documents:

Page	Description
DTD_FRAMESET	XHTML 1.0 Frameset Document Type Definition.
	The XHTML 1.0 Frameset DOCTYPE declaration is merged into the Web page when the tag <rdml MERGE="&amp;DTD_FRAMESET"&gt; is found. If you modify this page, don't include RDML tags.</rdml 
DTD_STRICT	XHTML 1.0 Strict Document Type Definition.
	The XHTML 1.0 Strict DOCTYPE declaration is merged into the Web page when the tag <rdml MERGE="&amp;DTD_STRICT"&gt; is found.</rdml 
	If you modify this page, don't include RDML tags.
DTD_TRANSITIONAL	XHTML 1.0 Transitional Document Type Definition.
	The XHTML 1.0 Transitional DOCTYPE declaration is merged into the Web page when the tag <rdml MERGE="&amp;DTD_TRANSITIONAL"&gt; is found. If you modify this page, don't include RDML tags.</rdml 

For more details, refer to LANSA for the Web XHTML.

# **6.9 Process Specific Page Components**

LANSA Web functions allow you to customize some of the shipped page components to be process specific. The following components can be defined as process specific:

- STDHEADER
- STDFOOTER
- STDBANNER

Whenever LANSA encounters a request for these components, it will search for a process specific version of the component. If the process specific component is found, it will automatically be used. If it cannot find a process specific component, it will then use the default component.

To create a process specific component, you must create a page Web component using the following naming convention:

<process name>\_STDHEADER

<process name>\_STDFOOTER

<process name>\_STDBANNER

where <process name> is the name of the LANSA process. For details of creating page Web components, refer to Page.

This approach in customizing the header, footer and banner components for your LANSA processes allow you to tailor individual processes, without having to edit any of the generated HTML. The customized headers and footers will be automatically used by LANSA Web functions, provided the components conform to the naming convention.



# 7. RDML Tags

LANSA automatically embeds special tags into the generated HTML pages for Web functions. These tags are used to dynamically build a required page when served to the client device. An understanding of the LANSA tags is very important if you are planning to modify the generated HTML pages. Review the following:

7.1 What are LANSA Tags? 7.2 How Do LANSA Tags Work? 7.3 LANSA Tags Example 7.4 Using <RDML> and </RDML> Tags 7.5 < RDML BUTTON> 7.6 < RDML CHECKVALUE> 7.7 < RDML COMPONENT> 7.8 < RDML COOKIES> 7.9 < RDML FUNCTION> 7.10 < RDML INCLUDE> 7.11 < RDML LAYOUT> 7.12 <RDML MERGE> 7.13 < RDML MESSAGES> 7.14 <RDML NOTCONDITION> 7.15 < RDML ONCONDITION> 7.16 < RDML ONMODE> 7.17 <RDML PAGE> 7.18 < RDML PARENT> 7.19 <RDML SETTMPFLD> 7.20 <RDML SSI> 7.21 <RDML TRACE> 7.22 Reserved Words



WEB007 - LANSA Tags

# 7.1 What are LANSA Tags?

LANSA Web functions have special tags which can be seen when editing the LANSA generated HTML. These tags are simply instructions to LANSA to perform certain tasks when creating the final version of the HTML page which will be transmitted to the client browser or other computing device.

LANSA tags are identified by a prefix of <RDML>. For example,

<RDML COMPONENT="STDHEADER">

These tags are automatically embedded as part of the generated HTML documents. LANSA tags can also be manually added by the developer.

These tags are not related in any way to LANSA RDML commands used in functions. The word "RDML" was chosen because it is easily identified as being LANSA related.

Once the generated HTML is processed by LANSA for the Web, these LANSA tags are removed. The tags do not appear in the completed HTML page.

If you intend to create LANSA Web components, a good understanding of the LANSA tags is essential as they will allow you to exploit the power of LANSA for the Web for your Web Function Applications.



WEB007 - LANSA Tags

# 7.2 How Do LANSA Tags Work?

Many LANSA tags are automatically embedded into the generated HTML for a Web function. For example, the inclusion of the standard header and footer components are achieved using LANSA tags. Input fields on the Web page and Message boxes are controlled by a LANSA tag. These tags are an important part of LANSA generated HTML.

A Web developer may choose to enhance the generated HTML using the Web Function Editor. The developer can modify the HTML to include more LANSA tags. For example, the developer might add a Web component to display a drop down box for an input field.

When the client or browser requests the LANSA Web function, the LANSA tags are decoded as the HTML is dynamically generated. LANSA will follow the instructions defined by the LANSA tags when creating the final HTML page. For example, the following LANSA tags:

```
Employee Number is <RDML MERGE="EMPNO">.
<br />
<RDML ONCONDITON="EMPNO">
The Employee number is not blank.
</RDML>
<RDML NOTCONDITON="EMPNO">
The Employee number is blank.
</RDML>
```

will tell LANSA to insert the required value for the #EMPNO field and will determine the appropriate text to display. If the EMPNO field has a value of A0001, then the final HTML sent to the browser will appear as follows:

Employee Number is A0001.

The Employee Number is not blank.

If the EMPNO field is blank, then the final HTML sent to the browser will appear as follows:

Employee Number is .

The Employee Number is blank

Notice that when the document is served to the browser or computing device, it contains no LANSA RDML tags. It is just HTML.



# 7.3 LANSA Tags Example

Consider the following example of LANSA generated HTML for a Web function:

Line

- 1. <html xmlns="http://www.w3.org/1999/xhtml">
- 2. <header><title>Enrol Employee</title></header>
- 3. <body bgcolor="white" background="
- <RDML MERGE="\*LW3CLNTBKGND">">
- 4. <form action="/CGI-BIN/WEBPAGE?FUNCTION+
- <RDML MERGE="&SESSION">" method="post">
- 5. <RDML COMPONENT="STDHEADER">
- 6. <center><h1><RDML MERGE="&FUNCTION"></h1></center>
- 7. <br />
- 8. <RDML MERGE="&MESSAGES">
- 9. <h3><RDML MERGE="EMPNO"></h3>

The LANSA tags include instructions for LANSA to:

- set the background to the image you have configured (line #3)
- set the session information (line #4)
- include the Standard Header page (line #5)
- display the Function description (line #6)
- display LANSA messages if there are any (line #8)
- display the multilingual description of the field (line #9).

It is important to remember that the tags only appear in the internal LANSA HTML documents. The following line of HTML:

<center><h1><RDML MERGE="&FUNCTION"></h1></center>

might appear as follows in the final HTML presented to the browser:

<center><h1>Enrol Employee<h1></center>

after the function name has been dynamically inserted into the HTML.



## 7.4 Using <RDML> and </RDML> Tags

The LANSA <RDML> tags can be used almost anywhere in your Web function HTML pages. For example:

```
<body background="<RDML MERGE="*LW3CLNTBKGND">">
```

will insert the background color (using the \*LW3CNTBKGND graphic variable) into the HTML tag.

If you have a LANSA <RDML> tag which has an associated </RDML> end tag, then the </RDML> tag must be used in a separate line and it must not contain any other LANSA tags.

The following is an example of the proper syntax with the </RDML> on a separate line:

```
<RDML BUTTON="&HELP">
```

```
<input type="image" name="&CANCEL" src="
```

<RDML MERGE="\*LW3IMGHELP">" />

```
</RDML>
```

#### Do NOT use the following types of statements:

<RDML BUTTON="&HELP">

<input type="image" name="&CANCEL" src="

```
<RDML MERGE="*LW3IMGHELP">" /></RDML>
```

As shown in the first example, you can include additional <RDML> tags within an <RDML> </RDML> pair, but you cannot embed an <RDML> tag within another tag.

#### **Do NOT use the following type of statements:**

<RDML COMPONENT="<RDML MERGE="EMPNO">">



#### 7.5 <RDML BUTTON>

Syntax: <RDML BUTTON="<button>"> </RDML>

Description: These tags are used to check the status of the specified button in the LANSA Web function. If the particular button is not enabled in the function, the lines encapsulated by these tags are ignored by LANSA for the Web. Otherwise, the lines are processed accordingly.

The <button> value can be one of the following 7.22 Reserved Words:

&EXIT

&CANCEL

&ADD

&DELETE

&CHANGE

&PROMPT

&USER1

&USER2

&USER3

&USER4

&USER5

There is a special value, &WEBEVENT, which is used in conjunction with WEBEVENT functions. In this case, if the function is a WEBEVENT function, the lines encapsulated by these tags are ignored.

Consider the following example:

```
<RDML BUTTON="&USER1">
<input type="image" src="/IMAGES/USER1.GIF" name="&USER1" />
<b>Click this button to search</b>
</RDML>
```

If the Web function has a user key1 specified in the DISPLAY or REQUEST command, (e.g. REQUEST FIELDS(#GROUP1) DESIGN(\*DOWN)

USER\_KEYS((01 'Search') ), then the statements within the start and end tags are executed. In this case, an image will be displayed for the button along with some text.

The <RDML BUTTON> tag should not be confused with the <RDML MERGE=&BUTTONS> tag. The merge tag is used to embed the active buttons into the HTML page whereas the <RDML BUTTON> tag is used to control whether or not a block of HTML lines will be executed.

### 7.6 <RDML CHECKVALUE>

Syntax: <RDML CHECK VALUE="YES"> </RDML>

Description: These tags instruct LANSA to compare the current field value with the value of each option in the Visual Web component. If the values match, the HTML line is modified to make that particular option selected.

In addition, these tags also instruct LANSA to modify the name of the component accordingly – whether it is in a WEBEVENT function or used in a browse list.

For example, a drop down Visual Web component might include the following:

```
<RDML CHECKVALUE="YES">
<select size="1" name="GENDER">
<option value="M">Male</option>
<option value="F">Female</option>
```

```
</select>
```

```
</RDML>
```

<RDML CHECKVALUE> is an instruction to LANSA to compare the current value of the field with the values indicated by the individual parts contained in the Visual Web component. If a matching value is found, LANSA modifies the HTML line to highlight the value selected.

In the above example, if the current value is "F", LANSA will modify the line containing the value and insert the HTML attribute selected="selected" in the corresponding <option> element. When the browser displays this Web component, the "Female" entry in the drop down is selected automatically.

### 7.7 <RDML COMPONENT>

Syntax: <RDML COMPONENT="<component>" MODE="<mode>">

Description: This tag instructs LANSA to include components into the HTML page being processed. It allows you to use the Web component technology provided by LANSA Web functions.

The MODE keyword allows you to specify the correct component to use according to the screen mode. The mode can be either "I" (input mode) or "O" (output mode). This keyword is used for mode dependent Web components and is optional.

LANSA Web functions allow you to dynamically define the component you want to use in conjunction with this tag. If you want to dynamically set the component name, the syntax of the tag is:

<RDML COMPONENT="&FLD\_<field name>">

where <field name> is the name of your field in the RDML function. This is the field that contains the name of the actual component to use with the tag. The value of the field should be set in the RDML function.

Once the <RDML COMPONENT> tag is embedded into the HTML, you can change the definition of the Web component without having to edit the function or to recompile the function. You can modify the contents of the Web component independently.

For example, you can change the Visual Web component type from a drop down to a set of radio buttons by just changing the definition of the Visual Web component. Once the change is made, it will be reflected dynamically when the Web component is next used. There is no recompiling or editing of the HTML required.

The following example instructs LANSA to include a standard HTML header in the page:

<RDML COMPONENT="STDHEADER">

The MODE keyword allows you to specify whether the component is used for input or output, i.e. is the user entering data or is data simply being displayed to the user. For example:

```
<RDML COMPONENT="DEPT" MODE="I">
```

indicates that a component called DEPT should be embedded when the page is used for entering data or input. This component might be a drop down list or a set of radio buttons which are used to enter a department code.

## **Dynamic Embedding of Components**

LANSA Web functions allow you to dynamically define the component you want to use in conjunction with the <RDML COMPONENT> tag. This is done by using the special "&FLD\_" designation with a field in your RDML function. For example:

<RDML COMPONENT="&FLD\_DEPTMENT">

In your RDML function, the DEPTMENT field will contain the name of the component which you want to embed in your application. For instance, if you were to CHANGE FIELD(#DEPTMENT) TO(ADM) in your function, then a component named ADM would be used. This is similar to using the following statement:

```
<RDML COMPONENT="ADM">
```

By using the &FLD\_ feature, you are able to set the component name in your function rather than coding the value into your HTML page.

### 7.8 <RDML COOKIES>

Syntax: <RDML COOKIES="<page name>">

Description: This tag allows you to set cookies in your application for your own specific purposes.

If you want to use this tag, this tag must precede the <RDML LAYOUT> tag in the page.

A cookie is a small piece of information which you can store on the client with a Web browser. At a later time, you can retrieve the information back from the browser. For example, cookies can be used by your application to identify the user. LANSA allows you to treat cookies as fields in your application. You can regard cookies as LANSA fields in your RDML function. Cookies are not used by the LANSA web function transaction server to maintain a persistent state in your application.

To set a cookie in your Web function, you will need to edit the HTML generated for the function to include the LANSA cookie tag:

```
<RDML COOKIES="&UDCOOKIES">
<RDML LAYOUT>
<!-- Process : PCTEST PC Test -->
<!-- Function : TESTFUN Test Function -->
<!-- Page : 001 -->
```

The <RDML COOKIES> tag should be one of the first lines on the page and it must not be indented. It must precede the <RDML LAYOUT> tag.

In this example, a page called DEFAULT\_UDCOOKIES is used. For more details about user defined pages, refer to User Defined Default Pages. This page can be created with the Web Function Editor and might contain the following statement:

```
USRID=<RDML MERGE="EMPNO">; expires=Fri, 23
Jun 2000 07:00:00 GMT
```

```
USRNM=<RDML MERGE="SURNAME">; expires=Fri, 23
Jun 2000 07:00:00 GMT
```

In this example, two cookies, USRID and USRNM, are set and will be stored on the client by the browser. The cookie USRID contains the value of the EMPNO field and the USRNM cookie contains the value of the SURNAME field.

Once you have set the cookies for your application, the cookies will be returned to any of your LANSA applications as LANSA fields. In the example above, if you want to read the value of the cookies, you will need to define fields in your RDML function for USRID and USRNM.

Cookies are created and stored with the CGI-BIN location as the name. When a process is setup with authentication a cookie will be created to correspond to the AUTHLIB location. For any subsequent requests, the cookies will revert to using the CGI-BIN location.

A client browser can be defined to not accept cookies. You must be careful about how you plan to use cookies in your application.

#### 7.9 <RDML FUNCTION>

Syntax: <RDML FUNCTION>

Description: This tag is used by the e-Business Framework Wizard in the layout pages and instructs LANSA to embed the HTML for your function at a particular point in the layout. Do not modify this tag.

For example, the DEFAULT\_LAYOUT pages might appear something like the following:

```
<RDML MERGE="&DTD_TRANSITIONAL">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title><RDML MERGE="&FUNCTION"></title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<RDML MERGE="&STYLE">
</head>
<RDML MERGE="&STYLE">
<RDML MERGE="&STYLE">
<RDML MERGE="&SCRIPT">
<body bgcolor="<RDML MERGE="*LW3CLNTCOLOR">" background="
```

```
<RDML MERGE="*LW3CLNTBKGND">">
<RDML FUNCTION>
</body>
</html>
```

### 7.10 <RDML INCLUDE>

Syntax: <RDML INCLUDE="<field name>">

Description: This tag can be used to include the contents of a physical file into the HTML output. The filename of the file to be included is given by the value of the field specified. This can be used as an alternative to the SSI #include instruction.

The filename given must be a full absolute path valid for the underlying operating system. On iSeries, the filename must be valid for the IFS.

The content of the file to be included should be in native encoding of the underlying operating system, for example, on iSeries, the file is normally encoded in EBCDIC.

Note that the user profile used to run LANSA for the Web needs to have sufficient privileges to read the file to be included.

If the file to be included does not exist or cannot be read, the message "404 Object Not Found" will be included instead.

For example, on iSeries, if a field named INCFILE contains the value "/temp/inc.html" and for the following HTML:

Some text before include<br />
<RDML INCLUDE="INCFILE">
Other text after include<br />

The content of the file "/temp/inc.html" on the IFS will be included between the two text only lines.

On Windows, the field should contain a value like "C:\Temp\inc.html".

### 7.11 <RDML LAYOUT>

Syntax: <RDML LAYOUT HEADER="<header page>">

Description: This tag instructs LANSA to use a layout page. It looks for a process specific layout page, <process name>\_LAYOUT. If it does not exist, the default layout page, DEFAULT\_LAYOUT, is used.

This tag is generated by LANSA.

If you want to use a specific header page with a particular function, you will need to edit the HTML page for the Web function and extend this tag to include the HEADER attribute. In this case, <header page> is the name of the page containing the header information for the function.

In your LANSA Web functions, this tag will most often appear as follows:

```
<RDML LAYOUT>
```

<!-- Process : IIPROC01 Test Process -->

<!-- Function : IIFN001 Display Sections -->

<!-- Page : 001 -->

Do NOT remove this tag as it is required if you are using the e-Business Framework Wizard.

If you wish to use a specific header page, you can use the following tag:

<RDML LAYOUT HEADER="header1">

where "header1" is the HTML page created with the Web Function Editor.

### 7.12 <RDML MERGE>

Syntax: <RDML MERGE="<field>" EDITCODEI="<value>" EDITWORDI="<value>" EDITCODEO="<value>" EDITWORDO="<value>">

Description: This tag instructs LANSA to merge specific fields or information into the document. The <field> can be:

- A field defined in your Web function.
- A system variable.
- A LANSA Web function reserved keyword. (For a list of reserved keywords, refer to 7.22 Reserved Words). Field names prefixed by either the '@' or '&' character are reserved.

The EDITCODE and EDITWORD attributes are automatically inserted by LANSA based on the field definitions in the LANSA repository. If input mode, EDITCODEI and EDITWORDI will be used. If output mode, then EDITCODEO and EDITWORDO will be used. Do not change these parameters.

For example, to set the background image for the HTML page you could use the following:

<body background="<RDML MERGE="\*LW3CLNTBKGND">">

To include the company LANSA system variable into an HTML heading, you would use:

```
<h1><RDML MERGE="*COMPANY"></h1>
```

To merge the field #EMPNO from your RDML function, you would use:

<RDML MERGE="EMPNO">

In multilingual applications, this token represents the description for a particular field:

<RDML MERGE="&T0001+0001+0034">

To include the function name using the &FUNCTION reserved word, you would use:

<RDML MERGE="&FUNCTION">

To include the workstation messages, you would use:

<RDML MERGE="&MESSAGES">

If there are any function keys which were not handled in the STDHEADER, this tag will cause them to be displayed as buttons on the Web page:

<RDML MERGE="&BUTTONS">

For details of using the MERGE tag for data apportionment, refer to Automatic Data Apportionment.

### 7.13 <RDML MESSAGES>

Syntax: <RDML MESSAGES>

Description: This tag instructs LANSA to use the line containing this tag repetitively, once for each message in the application. If you are not using a customized message presentation, you will not use this tag.

The RDML MESSAGES tag is used in a message presentation layout standard page such as DEFAULT\_MSGPRES or cessname>\_MSGPRES.

The line containing the <RDML MESSAGES> tag cannot contain any other LANSA tags.

For example, the following is a sample standard message layout which you might find in the DEFAULT\_MSGPRES page:

```
<img src="
<RDML MERGE="*LW3IMGMESSAGES">" alt="Messages" border="0"
/>
<b><RDML MESSAGES></b>
```

The <RDML MESSAGES> tag in the line is replaced by the message in the application.

The RDML MESSAGES tag should not be confused with the <RDML MERGE="&MESSAGES"> tag. This tag is used in your standard HTML pages. It indicates where LANSA should embed the message window.

### 7.14 <RDML NOTCONDITION>

Syntax: <RDML NOTCONDITION="<field>" VALUE="<value>"> </RDML>

Description: These tags are used by LANSA to determine if the lines encapsulated by these tags should be sent to the browser or not.

These tags allow you to programmatically set the contents of a field depending on certain logical conditions.

The <field> can be:

- A field in your LANSA function.
- A system variable.
- A reserved keyword.

If the field, <field>, does not exist or the contents of the field is blank or zero, then the lines are sent to the browser, provided the <value> parameter is not specified.

The *<*value*>* parameter is optional. If the *<*value*>* is specified, then the lines are only sent to the browser if the current field value does not match the value parameter.

There are reserved <field> values. These are special instructions for LANSA Web functions. These are:

• &DEBUG

Check if batch debugging is enabled in the initial URL request. If it is, the subsequent URL is modified to include the BDEBUG keyword before the request is submitted.

• &TASK

Check if a task identifier is specified in the initial URL request. If it is, the subsequent URL is modified to include the TASK\_ID keyword and the task identifier before the request is submitted.

• &USETMPFLD<index>

This is an instruction to check the HTML working fields identified by <index>. The <RDML SETTMPFLD> tag is used to set the values of these working fields. This specialized field is used to retrieve a particular working field. Do not use this keyword if you do not understand this description!

If the field is numeric and fieldname does not exist or the value is "0", then the lines encapsulated by the tags ARE sent to the browser.

If the field is alpha and fieldname does not exist or the value is "0" or is blank, then the lines encapsulated by the tags ARE sent to the browser.

Consider the following example:

```
<RDML NOTCONDITION="DISCOUNT">
<h2>You do not have a discount.</h2>
</RDML>
```

If the field DISCOUNT does not exist, then the user will see the message.

Let the DISCOUNT field be numeric. If in the RDML function the DISCOUNT field is 0, then the HTML will be used and the message will be displayed to the user. When the discount is not zero, the user will not see the message.

Let the DISCOUNT field be alpha. If in the RDML function the DISCOUNT field is blank, then the HTML will be used and the message will be displayed to the user. If the discount is "50 percent", then the user will not see the discount message displayed.

### 7.15 <RDML ONCONDITION>

Syntax: <RDML ONCONDITION="<field>" VALUE="<value>"> </RDML>

Description: These tags are used by LANSA to determine if the lines encapsulated by these tags should be sent to the browser or not.

These tags allow you to programmatically set the contents of a field depending on certain logical conditions.

The <field> can be:

- A field in your LANSA Web function.
- A system variable.
- A reserved keyword.

If the field, <field>, exists or the contents of the field is nonblank or not zero, then the lines are sent to the browser, provided the <value> parameter is not specified.

The *<*value*>* parameter is optional. If the *<*value*>* is specified, then the lines are only sent to the browser if the current field value matches the value parameter.

There are reserved <field> values. These are special instructions for LANSA Web functions. These are:

• &DEBUG

Check if batch debugging is enabled in the initial URL request. If it is, the subsequent URL is modified to include the BDEBUG keyword before the request is submitted.

• &TASK

Check if a task identifier is specified in the initial URL request. If it is, the subsequent URL is modified to include the TASK\_ID keyword and the task identifier before the request is submitted.

• &USETMPFLD<index>

This is an instruction to check the HTML working fields identified by <index>. The <RDML SETTMPFLD> tag is used to set the values of these working fields. This specialized field is used to retrieve a particular working field. Do not use this keyword if you do not understand this description.

If the field is numeric and fieldname does not exist or the value is "0", then the lines encapsulated by the tags ARE NOT sent to the browser.

If the field is alpha and fieldname does not exist or the value is "0" or is blank, then the lines encapsulated by the tags ARE NOT sent to the browser.

Consider the following example:

```
<RDML ONCONDITION="DISCOUNT">
<h2>Your discount amount is:</h2>
<RDML MERGE="DISCOUNT">
</RDML>
```

If the field DISCOUNT exists, then the user will see the message.

Let the DISCOUNT field be numeric. If in the RDML function the DISCOUNT field is "0", then the HTML will not be used and information about the discount will not be displayed to the user. When the discount is not zero, the user will see the discount amount displayed.

Let the DISCOUNT field be alpha. If in the RDML function the DISCOUNT field is blank, then the HTML will not be used and information about the discount will not be displayed to the user. If the discount is "50 percent", then the user will see the discount amount displayed.

The following example uses a value:

```
<RDML ONCONDITION="DISCOUNT" VALUE="50">
<h2>Your discount amount is 50.</h2>
<RDML MERGE="DISCOUNT">
</RDML>
```

### 7.16 <RDML ONMODE>

Syntax: <RDML ONMODE="<mode>"> </RDML>

Description: These tags are generated when you compile your functions that may have more than one mode of operation for a particular screen display. These tags instruct LANSA to carry out various screen instructions (the lines encapsulated by these tags) based on the current mode of operation.

The valid values for <mode> are:

- ADD Add
- CHG Change
- DLT Delete
- DSP Display

These tags are used internally by LANSA Web functions. You should not modify these tags.

The ONMODE tags are used in the HTML generated for a LANSA Web function when there is more than one mode of operation for a particular screen display. For example, an RDML DISPLAY statement may have ADD, CHANGE, DELETE and DISPLAY modes.

Consider the following example:

```
<h2>Employee Number: </h2>
<RDML ONMODE="ADD">
<input type="text" name="EMPNO" size="5" value="" />
</RDML>
<RDML ONMODE="DIS">
<RDML MERGE="EMPNO">
</RDML>
```

If the DISPLAY is in an ADD mode, the user will be expected to enter a value for EMPNO so the following HTML is sent to the user's browser:

<input type="text" name="EMPNO" size="5" value="" />

but when information is only being displayed to the user, the following

instruction is used:

<RDML MERGE="EMPNO">

which simply displays EMPNO as an output field in the user's browser.

#### 7.17 <RDML PAGE>

Syntax: <RDML PAGE="<page>">

Description: This tag instructs LANSA to include another page (identified by <page>) from the LANSA Web Repository.

This tag is useful when you need to create a process specific default page, but you still want to include the original default page itself.

For example, you may want to include an additional JavaScript function for the PSLSYS process, but still use all the other JavaScript functions shipped in DEFAULT\_SCRIPT. When you create the PSLSYS\_SCRIPT page, the DEFAULT\_SCRIPT page is no longer used. To overcome this problem, you use the RDML PAGE tag. Your PSLSYS\_SCRIPT page might appear as follows:

```
<script type="text/javascript" language="javascript">
//<![CDATA[
function handleTest()
{
.... your new function
}
//]]>
</script>
<RDML PAGE="DEFAULT_SCRIPT">
```

#### 7.18 <RDML PARENT>

Syntax: <RDML PARENT="<parent>">

Description: This tag is used by the e-Business Framework Wizard when you choose to adopt the layout of a particular LANSA process from another LANSA process. It will instruct LANSA to use the layout definition from the named process.

For more details, refer to the Adopt Layout - Technically Speaking in the *Web Functions Wizard Guide*.

When specifying the name of the parent for the LANSA process, do not use the same LANSA process name itself. If you do this, it will cause unpredictable results on your system.

The PARENT tag will have no impact if a process specific layout exists. The process specific components will be used over the adopted components in the parent specified.

This tag is commonly used in the process specific layout pages. For example, if PROC01 is going to adopt the layout of PROC02, then the following statement would appear in the PROC01\_LAYOUT page:

<RDML PARENT="PROC02">

#### 7.19 <RDML SETTMPFLD>

Syntax: <RDML SETTMPFLD="<index>" VALUE="<value>">

Description: These tags are used internally by the e-Business Framework Wizard. Do not alter these tags unless you are extremely familiar with the technical workings of the Wizard.

> LANSA provides you with 10 working fields at the HTML level. These fields are used to store temporary information that is evaluated when the HTML is processed.

The temporary information is only valid within a single HTML page request.

The <index> is base 0. To access the first working field, you need to use 0 as the <index> parameter.

These tags are intended for use by advanced Web developers only.

For example, you might see the following statements if you have used the e-Business Framework Wizard:

<RDML SETTMPFLD="10" VALUE="1"> <RDML SETTMPFLD="11" VALUE="1"> <RDML SETTMPFLD="12" VALUE="1">

These tags are used to control the menu components which appear in the layout.

#### 7.20 <RDML SSI>

Syntax: <RDML SSI="<state>">

Description: This tag is used to override the default setting for Server Side Includes (SSI) support as set by the LANSA for the Web Administrator.

The <state> can be either ON or OFF.

The main use of this tag is to enable SSI support at a function level. This is especially useful if you only require SSI support in a number of your functions.

If you want to use this tag, it must precede the <RDML LAYOUT> tag in the page.

<RDML SSI="ON">

<RDML COOKIES="&UDCOOKIES">

<RDML LAYOUT>

```
<!-- Process : PCTEST P.C. Test -->
```

```
<!-- Function : TESTFUN Test Function -->
```

```
<!-- Page : 001 -->
```

An example of a SSI instruction to launch a LANSA application is:

<!--#exec cgi="CGI-BIN/LANSAWEB?procfun+products+prodcat+web" -->

An example of an SSI instruction to include a static page is:

```
<!--#include virtual="/prdinfo.shtml" -->
```

#### 7.21 <RDML TRACE>

Syntax: <RDML TRACE>

Description: This tag is used to enable event logging. This tag is not generated by LANSA. It is recommended that if you wish to use event logging, then you should include this tag in your layout page.

> This tag is used to instruct LANSA to look up the trace page and determine which level of tracing is to be applied to your application. The appropriate tracing information will be written to the relevant files depending on the keywords specified in the trace file.

**Note:** Event Logging is only available when compiled with extended exchange enabled.

For more information review Event Logging.

# 7.22 Reserved Words

LANSA Web functions uses special keywords as part of the LANSA tag definitions. These keywords are prefixed by either the '@' character or the '&' character. Names prefixed by either the '@' or '&' character are reserved for internal use by LANSA.

The table below lists the major keywords used by LANSA Web functions. These keywords are generally used in conjunction with the <RDML MERGE> tag. (The keywords are also used with the RDML Button tag.) You can use any of these reserved keywords to merge any of these fields into your application.

Keyword	Description
&ADD	Display the Add key as a button.
&BL <list name=""></list>	Display the browse list identified by <list name="">.</list>
&BUTTONS	Display the buttons (function keys) of the function.
&BV_ <button></button>	Retrieve the description of the button specified in <
&CANCEL	Display the Cancel key as a button.
&CGI	Merge the path where the CGI interface program is LANSA for the Web configuration.
&CHANGE	Display the Change key as a button.
&CHECKNUMERIC	This is used in DEFAULT_SCRIPT page to conditi- used to validate numeric data in the browser
&DBCS	This is used in DEFAULT_SCRIPT page to conditiused to validate DBCS and SBCS data.
ⅅ <drop down="" name=""></drop>	Processes the field to be a drop down. Data for the Web function.
	This tag can be extended manually by using the SIZ
	The SIZE attribute allows you to visualize the drop SPLIT attributes allow you to manipulate the presei
	The example below is a list box for the DEPTMEN to apportion the first 4 characters as the VALUE to keyword is used, the description displayed in the lis

	<rdml &end"="" field="DEPTM&lt;/th&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DEBUG&lt;/td&gt;&lt;td&gt;Determine if batch debugging is on. If batch debug&lt;sub&gt;{&lt;/sub&gt; append the appropriate keywords.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DELETE&lt;/td&gt;&lt;td&gt;Display the Delete key as a button.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DEVICE&lt;/td&gt;&lt;td&gt;Merge the device name specified for batch debuggi&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DTD_FRAMESET&lt;/td&gt;&lt;td&gt;Merge the XHTML 1.0 Frameset DOCTYPE declar&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DTD_STRICT&lt;/td&gt;&lt;td&gt;Merge the XHTML 1.0 Strict DOCTYPE declaratic&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;DTD_TRANSITIONAL&lt;/td&gt;&lt;td&gt;Merge the XHTML 1.0 Transitional DOCTYPE dev&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&amp;END&lt;/td&gt;&lt;td&gt;The &lt;RDML MERGE=" merge="&amp;DDD01"> tag was introduc tag allows LWEB_JOB jobs to return to the pool of details, refer Using <rdml merge="&amp;END">.</rdml></rdml>
&EXIT	Display the Exit key as a button.
&FUNCHELP	Display the help text associated with the selected fi
&FUNCNAME	Embeds the current function name.
&FUNCTION	Display the function description.
&HEADER	Merge in the header information for the page. If the HEADER attribute is specified in the <rdmi associated with the attribute is used. If not, LANSA page. If it does not exist, then it will look for the de DEFAULT_HEADER.</rdmi 
&HELP	Display the Help key as a button.
&HIDDEN	Embed the hidden fields used by LANSA Web func internally and are defined in the DEFAULT_HIDD
&HMENU	Merge the Horizontal Menu component. This tag is e-Business Framework Wizard.
&IMAGE	Merge the location where the images are stored. Th configuration.
&JOBID	Merge the job identifier of the LANSA job. Note that this attribute must not be used as a unique

	if your applications consist of WEBEVENT functio WEBEVENT functions are restarted automatically jobs may be assigned to a different job.
&LINKDESC	Embeds Description of User Key in WEBEVENT f
&LINKFUNC	Embeds Linked Function Name of User Key in WE
&LINKPROC	Embeds Linked Process Name of User Key in WEE
&LMENU	Merge the Left Menu component. This tag is used i Business Framework Wizard.
&MESSAGES	Check if there are any LANSA messages. If there a
&MSGQ	Merge the message queue specified for batch debug
&PAGE	Embeds the current page identifier.
&PARTITION	Embeds the current LANSA partition.
&PARTLANG	Embeds the current LANSA partition language.
&PROCNAME	Embeds the current process name.
&PROMPT	Display the Prompt key as a button.
&RMENU	Merge the Right Menu component. This tag is used Business Framework Wizard.
&ROWNUM	This keyword must only be used in browse list com number of the browse list data.
	An optional attribute for this tag is the FORMAT ke format the row number. For example,
	<rdml <field="" format="4&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&lt;/td&gt;&lt;td&gt;instructs LANSA to format the row number as a 4-c if necessary.&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&lt;/td&gt;&lt;td&gt;Typically, this tag is used if you want to configure y appropriate name. In this case, the NAME attribute&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;&lt;/td&gt;&lt;td&gt;NAME=" merge="&amp;ROWNUM" name="">-</rdml>

&SCRIPT	Embed the JavaScript functions into the function. T DEFAULT_SCRIPT page.
&SESSION	Include the encoded session identifier.
&SESSPL	Include the encoded session identifier and the partit
&STYLE	Embeds the CSS page, if a CSS page exists.
&TASK	Merge the current task identifier.
&Tnnn+ssss+eeee	Retrieve the multilingual description for the field.
&TRACEID	This is used for event logging. When event logging allocated for each unique user to your site. This tag the trace identifier.
&UD <user defined="" page=""></user>	Embeds the user defined page. LANSA will initially identified by <process>_UD<user defined="" page="">. If the DEFAULT_UD<user defined="" page=""> page.</user></user></process>
&USER1	Displays the user defined key #1 as a button.
&USER2	Displays the user defined key #2 as a button.
&USER3	Displays the user defined key #3 as a button.
&USER4	Displays the user defined key #4 as a button.
&USER5	Displays the user defined key #5 as a button.

# 8. Graphic Variables

The e-Business Framework Wizard handles the common set of graphic variables. This means that you do not need to know the graphic variables in detail. For example, if you use the e-Business Framework Wizard, you do not need to know the name of the graphic variables. The Wizard displays the descriptive name of the variables. For more details, refer to 8.7 Graphic Variables and the e-Business Framework Wizard and refer to the *Web Functions Wizard Guide*. See System Wide Graphic Variables.

You only need to read this section if you want to have a detailed knowledge of LANSA graphic variables.

You may wish to review the following:

8.1 What are Graphic Variables?

8.2 Why Use Graphic Variables?

8.3 Types of Graphic Variables

8.4 Default Graphic Variables

8.5 Process Level Graphic Variables

8.6 Browse List Graphic Variables

8.7 Graphic Variables and the e-Business Framework Wizard

8.8 Technically Speaking



WEB006 - Graphic Variables

# 8.1 What are Graphic Variables?

LANSA for the Web uses special variables to store HTML settings so that you do not need to hard code information into your generated Web Function pages. These variables hold values for commonly used options like company logo or background images.

Graphic variables can store a wide range of settings for the HTML pages generated. By storing information in graphic variables, it makes HTML dynamic and easy to maintain. The values associated with graphic variables can be changed dynamically without having to either edit the HTML or recompile the function.

Developers can define their own graphic variables and use them in their LANSA HTML pages. The Web Function Editor is used to define graphic variables.

Graphic variables are used with the <RDML MERGE> tag. Refer to <RDML MERGE> for details of the LANSA tags.

LANSA for the Web uses LANSA system variables to store the graphic variables. Consequently, LANSA for the Web graphic variables exist at the LANSA system level. If you add or change a graphic variable, it can be used in all LANSA partitions.

## 8.2 Why Use Graphic Variables?

Imagine defining a background image which you want to include in your HTML Web Function pages. In each of the HTML documents you could code the following:

```
<body background="lansa.gif">
```

Now imagine that you wish to change the background to a new image file. You would manually have to edit each Web Function page to make the change.

However, using a LANSA for the Web graphic variable and the LANSA RDML tags, you can make this change without editing any HTML documents. The LANSA Web Function application can include the following HTML statement in each of the LANSA documents:

```
<body background="<RDML MERGE="*LW3CLNTBKGND">">
```

where \*LW3CLNTBKGND is set to "image.gif" to start.

To change to a new background, you simply change the value of \*LW3CLNTBKGND to "newimage.gif". Immediately, all pages served will have the new background. No other changes are needed. You do not need to recompile any Web functions. You do not need to edit the HTML.



# 8.3 Types of Graphic Variables

LANSA for the Web supports the following types of graphic variables:

- 8.3.1 Image File Graphic Variables
- 8.3.2 Color Graphic Variables
- 8.3.3 Text Graphic Variables

Image File variables allow you to associate the name of an image file with the variable. The variable simply stores the name of an image file you wish to use.

Color variables allow you to customize the background and font color or other colors set in your HTML attributes. For example, you would associate the background color with the \*LW3CLNTCOLOR graphic variable.

A Text variable allows you to associate text with a variable. This text could be used in any number of ways depending on where you want to dynamically define HTML and its attributes. For example, it could be used to set font sizes or types.



# 8.3.1 Image File Graphic Variables

An Image File variable allows you to associate an image with the variable. This variable is useful for displaying images on your applications. You can dynamically change the image associated with the variable, without having to edit the HTML or recompile the LANSA function.

If the graphic variable is an image file variable, the image associated with the variable must be stored in the Image Location as configured in LANSA for the Web using the LANSA for the Web Administrator.

The advantage of using graphic variables is that you can dynamically change the image associated with the variable, without having to edit the HTML or recompile the LANSA Web function. For example, you can customize settings like the company logo. Instead of embedding the company logo into every HTML page, you can define a graphic variable for the company logo. If the company logo is changed, the change needs only be done once, by redefining the graphic variable.

In the example below, the <RDML MERGE> tag is used to embed an image defined by the \*LW3IMGHOME variable, which is the image used for the Home button.

```
<
input type="image" name="&HOME" src="
<RDML MERGE="*LW3IMGHOME">" />
<h5 align="center">Home</h5>
```

LANSA for the Web allows you to customize the background color or the background image. You would associate the background color with the \*LW3CLNTCOLOR graphic variable, and the background image with the \*LW3CLNTBKGND graphic variable.

**Note:** If both graphic variables are defined in your system, the background image would take precedence.



## 8.3.2 Color Graphic Variables

You can define a graphic variable and associate a color setting with the variable. As an example, you can change the color of the Menu frame by changing the color setting associated with the \*LW3MENUCOLOR variable.

Graphic variables simply store the values as text. For color settings, the values can be entered as hexadecimal values in RRGGBB (red green blue) or as the color name. For example, you could enter the value "#FF0000" or the value "RED".

LANSA for the Web also allows you to define the color settings used in browse lists. For more details, refer to 8.6 Browse List Graphic Variables.



## 8.3.3 Text Graphic Variables

This type of variable allows you to associate a string of text with a variable. You can use the text graphic variables anywhere that you want a variable in your HTML. The variable can hold a string of characters up to 255 characters long. The text variable could contain a word, a number, a phrase, or even a complete HTML string. The text associated with such a variable can be changed dynamically without having to edit the HTML or recompiling the function.

For example, you could create a graphic variable called \*LW3FONTFACE which would store the name of the font you wish to use in your HTML. You might give it a value of "COURIER". A graphic variable called \*LW3FONTSIZE could be used to control the font size. You might give it a value of "12". These variables could be used as follows:

```
<font face="<RDML MERGE="*LW3FONTFACE">" size="
<RDML MERGE="*LW3FONTSIZE">">
```

You could also create a variable called \*LW3COPYRIGHT. This variable could be used to display the copyright information on your pages. For example, the variable might be set to the following HTML string:

"<strong> All images and text are copyrighted by XYZ Inc. 2000</strong>"

The power of graphic variables is the ability to change these values dynamically.

Imagine that you have multiple Web sites around the world. These sites are mirror sites so your applications are identical. If you wanted to indicate to the user which site they were actually using, you could create a graphic variable called \*LW3WEBSITE. The variable would have a different value on each machine. It might have a value of "CANADA" or "AUSTRALIA". The HTML would be identical. It would simply read:

<b>You are accessing <RDML MERGE="\*LW3WEBSITE"> </b><br/>On the Canadian mirror site, it would appear as:

<b>You are accessing CANADA </b>



## **8.4 Default Graphic Variables**

LANSA for the Web provides a default set of graphic variables for use with your Web functions. The following list of variables does not include the process-specific or list-specific variables.

#### **Default Graphic Variable Description**

*LW3CPYLOGO	Company logo.
*LW3IMGCANCEL	Image file for the Cancel button.
*LW3IMGFBORDER	Image file for the Standard Footer border.
*LW3IMGHBORDER	Image file for the Standard Header border.
*LW3IMGHELP	Image file for the Help button.
*LW3IMGHOME	Image file for the Home button.
*LW3IMGLANSA	Image file for the LANSA logo.
*LW3IMGMENUSELECT	Selection image for process menu items.
*LW3IMGMESSAGES	Image used in conjunction with LANSA messages.
*LW3IMGMSGS	Image file for the Messages button.
*LW3IMGOK	Image file for the OK button.
*LW3MENUCOLOR	Background color for the Menu area. If no value is specified, the background defaults to gray (#COCOCO).
*LW3SESSIONID	Encoded job identifier. This is used in LANSA tags.
*WEBIPADDR	Returns the IP address of the current user.
*WEBMODE	Returns a 'Y' if the LANSA application is running under Web enabled mode.
*WEBPATHINFO	Returns the value attached to the PATH_INFO Web server environment variable.

*WEBREFERRER	Returns the value attached to the HTTP_REFERER Web server environment variable.
*WEBSCRIPTNAME	Returns the value attached to the SCRIPT_NAME Web server environment variable.
*WEBUSER	Web Server/400 or Internet Connection Server for OS/400 user profile.

**Reminder:** Graphic Variables are defined at the LANSA system level. They are shared by all partitions.

### **8.5 Process Level Graphic Variables**

LANSA for the Web allows you to customize the graphic variables used for a specific process and its functions. For example, you can customize the background image used by the PSLSYS process by creating a \*LW3PBGI\_PSLSYS graphic variable. This variable will be used instead of the \*LW3CLNTBKGND variable.

#### **Process Customization**

Default Graphic Variable	Specific Graphic Variable	Description
*LW3CLNTBKGND	*LW3PBGI_ <process name&gt;</process 	Background image for the Client (body) area.
		A default image is provided by LANSA for the Web.
		LANSA for the Web allows you to either set the background image or the background color (LW3CLNTCOLOR). If both variables are set, the background image takes precedence.
*LW3CLNTCOLOR	*LW3PBGC_ <process name&gt;</process 	Default background color for the Client area. If no value is associated with the client background (*LW3CLNKBKGND), the value associated with this variable will be used to set the background color. The background image takes precedence over the background color

setting.

*LW3IMGMENUSELECT	*LW3PMSI_ <process name&gt;</process 	Image used for the menu items in the process menu.
*LW3MENUSEP	*LW3PMSP_ <process name&gt;</process 	Image for menu separator. This image is used when the e- Business Framework Wizard builds the Menu components.
*LW3RMENUBKGND	*LW3RMBG_ <process name&gt;</process 	Color setting for the Right Menu component.

#### Tip

To have a Default Background image (\*LW3CLNTBKGND) and use a specific background color for a process, you could define the background image for this process to be empty (\*LW3PBGL).

## **8.6 Browse List Graphic Variables**

LANSA for the Web allows you to customize browse lists in your Web function by using graphic variables. It is important to note that the HTML generated for browse lists is controlled by LANSA for the Web and cannot be manually edited. The use of graphic variables is an important method for customizing the presentation of the browse lists in Web Functions.

If a specific browse list graphic variable exists, LANSA for the Web will use it. Otherwise, it will use the default graphic variable. For example, you may wish to customize the background colors for alternate rows used for a browse list named EMPLIST. In this case, you will need to create a specific graphic variable \*LW3BLACB\_EMPLIST. This variable is used instead of the \*LW3BLACELLBCOLOR variable.

**Reminder:** Graphic Variables are defined at the LANSA system level, including specific graphic variables. When you create a graphic variable for a specific browse list name, it will impact all browse lists with that name in all partitions.

#### **HTML Browse List Customization**

Default Graphic Variable	Specific Graphic Variable	Description
*LW3BLACELLBCOLOR	*LW3BLACB_ <list name&gt;</list 	Background color for alternate rows in browse lists.
		If you want to have a transparent background, specify *NONE as the value of this variable.
*LW3BLACELLFCOLOR	*LW3BLACF_ <list name&gt;</list 	Foreground color for alternate rows in browse lists.
*LW3BLCELLBCOLOR	*LW3BLCCB_ <list name&gt;</list 	Background color for entries in browse lists. If you want to have a transparent background, specify *NONE as the value of this variable.

*LW3BLCELLFCOLOR	*LW3BLCCF_ <list name&gt;</list 	Foreground color for entries in browse lists.
*LW3BRWLSTMISC	*LW3BLMS_ <list name&gt;</list 	Miscellaneous attributes to be used in conjunction with the  tag for setting up a browse list. The attributes defined for this variable will be appended as attributes of the tag for browse list. You can override the
		default setting of the border, cellpadding and cellspacing attributes by using this variable.
*LW3BRWLSTSELECT	*LW3BL_ <list name&gt;</list 	The contents of this variable allows you to disable:
		<ul> <li>Borders around the browse list.</li> </ul>
		<ul><li>Column headings.</li><li>Selection image.</li></ul>
		<ul> <li>Borders around empty cells in the browse list.</li> </ul>
		To turn off any of the above features, you will need to define the variable as a Text variable. The content of the variable is then set to be:
		*Noxxxxx where xxxxx can be up to
		five characters, identifying the feature to disable.
		If you want to disable the border around the browse

list, specify a 'B' character. If you want to disable the column heading in the browse list, specify a 'C' character.

If you want to disable the selection image, specify an 'I' character.

If you do not want a HTML tag around your browse list, specify a 'T' character.

If you do not want borders displayed around empty cells in your browse list, specify a 'P' character.

Note that if you disable the selection image, none of the columns in the browse list will be hyperlinked, even though you have enabled LANSA for the Web to allow selection from any column in a browse list. For example, \*NOB would switch borders off.

*LW3BRWLSTSELIMAGE	*LW3BLI_ <list name&gt;</list 	Default selection image for browse lists.
*LW3COLHDGBCOLOR	*LW3BLBC_ <list name&gt;</list 	Background color for column headings for browse lists. If you want to have a transparent background, specify *NONE as the

		value of this variable.
*LW3COLHDGFCOLOR	*LW3BLFC_ <list name&gt;</list 	Foreground color for column headings for browse lists.

**Note:** LANSA for the Web disables the selection image column in browse lists for WEBEVENT functions. This is because WEBEVENT functions terminate as soon as the display command is processed. This means that the function is no longer available to process your selection.

## 8.7 Graphic Variables and the e-Business Framework Wizard

Instead of using the PC-based LANSA Web Utilities to modify the graphic variables and components, you can make these modifications using the e-Business Framework Wizard.

This picture shows the changes you can make to graphic variables using the ebusiness frameworks.

	Maintain WIZ_LIST2 Browse List			
	WARNING: Any modifications to this name in all partitions	a browse list variable w	vill apply to a	ll lists with
	Apply	Preview		
Home	Browse List Colors			Taa   Setto
Define Process Group	Background color for odd numbered rows	white	Color	Color Picker
	Foreground color for odd numbered rows	black	Color	Color Picker
File Search	Background color for even numbered rows	silver	Color	Color Picker
Help	Foreground color for even numbered rows	black	Color	Color Picker
anguage:	Column Headings			<u>Top   Botto</u>
ENG	Background color for column headings	seagreen	Color	Color Picker
Characteristics to modify: BROWSE	Foreground color for column headings	white	Color	Color Picker
Current Process:	Images			<u>100   10110</u>
current Flocess.	Selection image for this browse List	ballshad.gif	Image	
	Miscellaneous Attributes	-		<u>100   50110</u>
	HTML Attributes used in conjunction with the <	table> tag cellpadding="1" a	cellspacing="1"	
	Presentation Attributes			<u>Top   Botto</u>
	🔽 Table Borders			
	🗆 Column Headings			
	Selection Image			
	Table Tags			

The e-Business Framework Wizard simplifies the development of your Web applications.

Using the e-Business Framework Wizard, you do not need to know the exact name of the graphic variable of the associated browse list attribute you want to change. All possible attributes of a browse list are presented to you in a single screen so that you can quickly and easily identify what you want or need to change.

For more details about the e-Business Framework Wizard, refer to the *Web Functions Wizard Guide*. See System Wide Graphic Variables

# 8.8 Technically Speaking

LANSA for the Web graphic variables are based upon LANSA system variables. When you create a graphic variable using the Web Function Editor, a LANSA system variable is automatically defined. Because graphic variables are defined as system variables, they are system wide in LANSA. All partitions in the LANSA system will use the same set of graphic variables.

**Note:** When you import or export your application, you are moving system level variables! You will impact all partitions in your LANSA system.

A LANSA for the Web graphic variable must be defined as:

- a static (STATIC) alphanumeric (ALPHA) system variable in LANSA
- a variable with a length of 255 characters (decimals = 0)
- a variable which is evaluated by the 3GL program, W3@P2100.

## 9. Web Components

LANSA Web Functions uses Web component technology to build more powerful applications. It is very important that you have a good understanding of the Web component technology if you are building LANSA Web Function Applications. Review the following:

9.1 Introduction to Web Components

9.2 Manually Defined Web Components

9.3 Generated Web Components



WEB008 - Web Components

## 9.1 Introduction to Web Components

In order to understand what Web components are and how they are used, review the following:

- 9.1.1 What are Web Components?
- 9.1.2 Web Component Architecture
- 9.1.3 Web Component Example
- 9.1.4 Types of Web Components
- 9.1.5 Web Components and Modes
- 9.1.6 Automatic Embedding of Web Components
- 9.1.7 Dynamically Embedding Web Components
- 9.1.8 Considerations for Using Web Components

## 9.1.1 What are Web Components?

Web component technology is a powerful way of enhancing your LANSA Web Function applications. It allows you to create individual components (small portions or modules of an application), that can be embedded and re-used in your Web Function application.

For example, you can create a Web component, which is:

- A drop down list that displays State or Province codes,
- A single image or a group of buttons,
- A static HTML page,
- A complete menu with links to all of your applications.

You can create a Web component for almost any part of your application.

The Web component technology greatly simplifies the HTML pages and significantly reduces the maintenance efforts because components are centrally defined once and are automatically reused. Web components are an extension to the basic LANSA architecture and can be thought of as a repository for building Web Function applications. Components can be updated and generated automatically. Changes can be made to components without having to recompile RDML functions or edit HTML pages. The Web component technology in LANSA Web Functions allows you to easily customize your HTML pages.

The e-Business Framework Wizard uses Web components extensively.

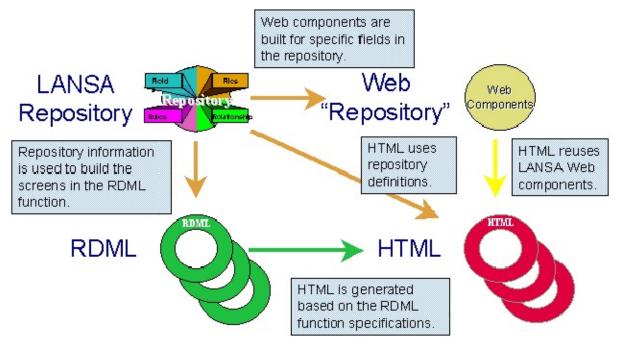
LANSA for the Web supports the following types of Web components:

- File
- Banner
- Page
- Script
- Text
- Web Link
- Visual

The type of LANSA Web component determines how LANSA handles the component.

LANSA Web components are used in conjunction with the <RDML COMPONENT> tag. This tag instructs LANSA to process the Web component indicated by the tag. For more details, refer to <<u>RDML COMPONENT</u>>.

## 9.1.2 Web Component Architecture



The Web component technology is really an extension of the LANSA Repository architecture. The Web component registry acts like a Web repository. It centrally defines and stores components which are used in the Web Function HTML pages.

The LANSA Repository centrally stores much of the information about the business application. This information is used by LANSA RDML functions. For example, screen layouts for RDML functions are based on the field definitions stored in the repository.

In much the same way, the LANSA Repository is used by the LANSA HTML pages which are generated from the RDML functions. The LANSA RDML function is used as the base for the pages while additional information for the HTML definitions are read from the LANSA Repository.

The LANSA Repository also interacts with the Web repository or Web component registry. For example, if a field has been defined with a drop down GUI characteristic, a Web component is automatically generated for the field and used in the page.

When the HTML is generated for a Web function, LANSA tags are used to embed the Web components from the registry. The Web components can be automatically added to the pages based on the field and component naming. For example, if a field DEPTMENT has a corresponding Web component DEPTMENT defined, this component will automatically be included into the HTML pages. With the Web repository/registry, the field will automatically have the same representation on all HTML pages. And if the representation is changed in the Web component registry, it will immediately be changed for all pages since the LANSA HTML is dynamically generated.

A key advantage of the Web component technology is that once the Web component is embedded into the generated page, the definition of the Web component can be changed without having to recompile the function or to edit the HTML page.

Many Web components will be manually created by the developer and then added to the HTML pages.

# 9.1.3 Web Component Example

Typically, an HTML page generated by LANSA consists of a number of Web components. The STDHEADER and STDFOOTER pages are regarded as LANSA Web components. These Web components are embedded into every function generated by LANSA. The STDHEADER includes a company logo and set of pushbuttons.

Web component technology allows you to modify an individual Web component without having to modify every application which requires the Web component. For example, you can modify the STDHEADER page and the changes will be reflected in every application which uses the STDHEADER Web component.

Once the change is made to the STDHEADER, it is immediately available to be used in all your HTML pages. RDML functions do not need to be recompiled. HTML pages do not need to be edited. It is a very fast and easy way of maintaining your Web Function applications.

For more details, refer to Standard HTML Page Components.



WEB005 - LANSA Process Pages and WEB008 - Web Components

## **9.1.4 Types of Web Components**

LANSA for the Web supports the following types of Web components for use with Web Functions:

Manually Defined Components:

- 9.2.1 Banner
- 9.2.2 Text
- 9.2.3 Web Link
- 9.2.4 Page
- 9.2.5 Script

Generated components:

- 9.3.1 Visual Web Component (Check Box , Drop Down, List Box, Radio Buttons).
- 9.3.4 File Web Component

The type of component determines how LANSA handles the Web component.

Manually defined Web components required the component definitions to be entered by the developer. For example, the developer must enter the JavaScript used in the Script component.

Generated Web components have their component definitions automatically created by LANSA for the Web. For example, LANSA for the Web will automatically generate the HTML for a Visual drop down component.

## 9.1.5 Web Components and Modes

LANSA for the Web allows you to define the following modes for Web components used in Web functions:

- Input
- Output
- Not Applicable.

The input mode components are used when a REQUEST or DISPLAY screen is input capable.

The output mode components are used when a REQUEST or DISPLAY screen is output only. The output mode component is always used for hidden fields, even on input capable screens.

The not applicable mode is used for components not used in the input or output screen area. For example a STDFOOTER or STDHEADER page is defined as not applicable since it is not part of the input or output screen area. The component is used to define the structure of the page.

When you compile your LANSA Web function, LANSA checks if mode dependent components exist for the fields defined in your function. If the mode of operation is input capable, it will use the input mode Web component, if one exists, to replace the field. If the mode of operation is output, it will use the output mode Web component, if one exists, to replace the field.

```
If mode dependent Web components are used in your Web function applications, these can be identified by the LANSA tag, <RDML COMPONENT="<field name>" MODE="<mode>">.
```

For example, if you have the following RDML statement in your Web function:

REQUEST FIELDS((#STDNEXT \*HIDDEN)(#DEPTMENT) (DEPTDESC \*OUTPUT))

and you have create Web components STDNEXT, DEPTMENT and DEPTDESC, then the following Web components will appear in your HTML page:

```
<RDML COMPONENT="STDNEXT " MODE="O">
<RDML COMPONENT="DEPTMENT " MODE="I">
```

<RDML COMPONENT="DEPTDESC " MODE="O">

The mode of the component is part of the component definition. For more details, refer to the specific type of Web component you wish to create.



## 9.1.6 Automatic Embedding of Web Components

LANSA Web components are used in conjunction with the <RDML COMPONENT> tag. This tag instructs LANSA to process the Web component indicated by the tag.

When a LANSA Web function is compiled, LANSA will check each field used in the REQUEST or DISPLAY and it will automatically substitute a Web component for the field, if a component exists. The Web component must also be defined with the correct mode.

For example, the Web function displaying the DEPTMENT field in a REQUEST statement would normally contain the following LANSA generated HTML tag:

<RDML MERGE="DEPTMENT ">

The MERGE simply inserts the field. However, once an input mode Web component named DEPTMENT is created for the DEPTMENT field, the LANSA generated HTML will automatically contain the following tag when the function is recompiled:

<RDML COMPONENT="DEPTMENT " MODE="I">

LANSA will embed the proper mode for the Web component. If there were no input mode DEPTMENT Web component, then a MERGE tag would be used.

You must recompile your Web functions if you want LANSA to automatically embed the Web components into the HTML pages. If you create Web components after you have compiled your functions, you must recompile the functions to embed the Web components, or you may manually edit the HTML to include the <RDML COMPONENT> tags if you do not wish to recompile your functions.

You can include \*HIDDEN fields in your displays to add Web components to your Web functions. All \*HIDDEN fields are considered output mode components.

You may also manually embed Web components by adding <RDML COMPONENT> tags to your HTML pages.

For more details, refer to <<u>RDML COMPONENT</u>>.



## 9.1.7 Dynamically Embedding Web Components

You can embed Web components dynamically in your Web function application by using the RDML COMPONENT tag with the &FLD option. In most situations, you will see the RDML COMPONENT tag used in LANSA HTML with the component name explicitly specified. However, if you want to set the name of the component when the function is executing, you can dynamically set the component name.

For example, if you have an HTML page which displays product information, the product displayed in the page is dependent on the product requested by the user. In such a page, the information is dynamic. It depends on the product selected when the function is executing.

In your Web function, you would need a field for the product component. It might be called PRODUCT and would be defined as an alpha field. In the HTML page, you would include the following line for the product component:

<RDML COMPONENT="&FLD\_PRODUCT">

This line instructs LANSA to use the contents of the PRODUCT field as the name of the Web component to embed.

In your Web function, you can dynamically set the value of the PRODUCT field according to the user request.

CHANGE FIELD(#PRODUCT) TO("ABC123")

Using the Web Function Editor, you would create a component named ABC123. This component might include some text and images for the specific product.

## 9.1.8 Considerations for Using Web Components

Following are some important considerations when using Web components with Web functions:

- LANSA Web components are used in conjunction with the <RDML COMPONENT> tag. This tag instructs LANSA to process the Web component indicated by the tag.
- If you use the NAME keyword when creating your HTML Web component, do not use names longer than 10 characters. Some Browsers (e.g. Netscape) may cause the LWEB\_JOB to fail by sending invalid data to the stack.
- Web components can be automatically embedded into an HTML page when the RDML function is compiled. Components are embedded based on the field name and the mode of the REQUEST or DISPLAY. For more details, refer to 9.1.6 Automatic Embedding of Web Components.
- Web components will be embedded for hidden fields in a display. The Web component will be defined as output mode for hidden fields. Defining hidden fields is a good method of embedding Web components.
- If you create a new Web component after your RDML functions have been compiled, you can manually edit the HTML page to use the <RDML COMPONENT> tag, or you can recompile your functions to automatically embed the components.
- You can change the definition of a component without recompiling your Web functions or editing your HTML pages.
- A LANSA Web component can be embedded into another LANSA Web component, provided they are not embedded recursively.
- You can embed Web components dynamically in your application by using the RDML COMPONENT tag with the &FLD option. For more details, refer to 9.1.7 Dynamically Embedding Web Components.
- LANSA Web components are created using the Web Function Editor. For more details, refer to Components Menu.
- Web components are defined at the partition level.
- Web component names must be unique in a partition. You may only duplicate a Web component name when there is a mode associated with the component. For example, you can create three components named DEPTMENT one for Input mode, one for Output mode and one as Not Applicable.

• If you have manually created HTML pages using the Web Function Editor, these pages should be registered as Page components, especially if you need to export these pages with your system. (You should register the DEFAULT\_xxxxx pages as components if you need to export these pages. Refer to for information about Web Application Deployment.



## 9.2 Manually Defined Web Components

LANSA for the Web supports the following Manually Defined Components for use with Web functions:

9.2.1 Banner
9.2.2 Text
9.2.3 Web Link
9.2.4 Page
9.2.5 Script.
Manually defined Web c

Manually defined Web components required the component definitions to be entered by the developer. For example, the developer must manually code the JavaScript used in the Script component. If you are using page or script component, you should refer to 9.2.6 Naming Page and Script Web Components.

You use the Web Function Editor to create Web components.



#### 9.2.1 Banner

A banner Web component allows you to insert advertisement banners into your Web function applications. Banners are a collection of images that are displayed one at a time. These images have an associated URL, usually the URL of the supplier of the advertisement.

A banner Web component allows you to define a banner once and to embed the advertisement into any of your applications. A number of the layouts in the e-Business Framework use a standard banner component, STDBANNER, in its schema.

LANSA for the Web takes care of the cycling of the images associated with the banner. The images of the banner have an associated sequence number. LANSA for the Web will cycle through the images in the banner Web component sequentially.

The banner Web component can be composed of other Web components. This allows you to dynamically change the layout of your pages, if you require such functionality.

The banner Web component is defined completely within LANSA for the Web, that is, you do not have to define any HTML to use this component.

When defining the banner Web component as a list of Web components, the Web components can consist of any type of Web components. For example, you could create a banner Web component that consists of a Visual drop down Web component, check box Web component, and radio button Web component. In addition, the banner Web component can embed another banner Web component. This allows you to customize the presentation of your data. A banner Web component can be embedded into any HTML page in your application.

For details about creating banner components, refer to Banner Component (HTML mode).



WEB008 - Web Components

#### 9.2.2 Text

A text Web component allows you to embed a piece of text into your Web function application. The text could be an HTML string or any other text. The text component is very similar to the text graphic variable, but the text component has the advantage of being defined at the partition level instead of the system level.

Using a text component allows you to define a string that can be manipulated without having to edit HTML pages or recompile functions. For example, you could create a text component called NEXTUPDATE. This variable could be used to display the date that you plan to do your next update on your pages. You might include this component in the STDFOOTER. The text component might have the following value:

<strong>Next update of these pages will be 2001-12-31.</strong>

Using the NEXTUPDATE component, you can change the date of your next update on all you Web pages with editing any HTML.

The length of the text is limited to 255 characters and the text entered may include HTML tags.

**Note:** RDML tags as part of the text are not resolved.

For details about creating text components, refer to Text Component.



## 9.2.3 Web Link

Web link components are only used with WEBEVENT functions. Web link components define the link to other functions for your WEBEVENT functions. You can use Web link components to link to other functions, instead of using the Keywords command in the Web Function Editor.

Web link components allow you to display the links as images instead of buttons. These components allow you to dynamically change the links as well as the presentation of the links, without having to recompile your WEBEVENT functions since you do not have to specify the USER\_KEYS parameter in the REQUEST or DISPLAY commands.

For details about creating Web link components, refer to Web Link Component.



## 9.2.4 Page

Page Web components are the most common type of component used by Web Function applications. A page Web component can be any block of HTML that you wish to re-use or make independent of the RDML function. Once the HTML is contained in a page component, it can be altered without requiring the Web function to be recompiled. Page Web components are useful if you want to embed some standard HTML into one or more of your applications.

The HTML of the component, is created by editing and saving a file, using the Web Function Editor. The saved file can then be registered as a Page Web component.

For example, the STDHEADER is a Page Web component. Notice that your page can use LANSA tags and can even include LANSA components. (Be sure that the components are not called recursively!)

For details about creating page Web components, refer to 9.2.6 Naming Page and Script Web Components.



WEB008 - Web Components

## 9.2.5 Script

A Script Web component allows you to create a JavaScript or VBScript function (or script fragment) and embed such a function into your Web function application.

The script file is defined using the Web Function Editor. LANSA for the Web does not validate the script functions specified in the script Web component. Once the file is defined in the Web Function Editor, it must be registered as a Web component.

LANSA for the Web does not restrict you to a particular script language. In other words, you could be creating a JavaScript or VBScript script Web Component. Again, LANSA for the Web does not validate the script functions specified in the script Web component. You must ensure that the script you have created is syntactically correct. You must verify your script before creating it as a script Web component.

One difference between a Script Web component and other Web components is that the last line of a Script Web component doesn't include a carriage return (or carriage return/line feed pair on Windows) at the end of the component. This prevents an unwanted line break when the component content is just a script fragment, rather than a complete script function. If you want the last line of your Script web component to end with a carriage return, insert a blank line at the end of the component.

If you are defining script functions, you should remember that these functions should be included in the header of the HTML pages so that the functions are loaded before they are called.

The JavaScript functions used by LANSA for the Web are contained in the DEFAULT\_SCRIPT component. You can also modify this component to include your own JavaScript functions, and then save the file as a process-specific page. Using this technique, you do not need to register the process-specific page as a Web page component.

For details about creating script Web components, refer to 9.2.6 Naming Page and Script Web Components and Script Component (HTML mode).

## 9.2.6 Naming Page and Script Web Components

When you create a Page or Script Web component, you must complete two steps:

1.Create the file with the HTML or JavaScript.

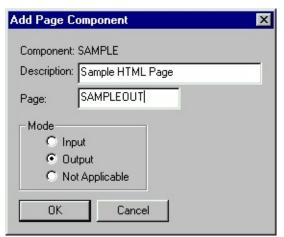
2.Register the file as a Page or Script component.

Using the Web Function Editor, you will need to create a new page to store the code (HTML or JavaScript) for the component. Generally, you would make the page name the same as the component name; however, these names do not need to be the same. If you have more than one component with the same name, because of input and output modes, you may want to define a naming convention to identify the appropriate pages.

For example, an HTML page called SAMPLEOUT can be created and saved using the Web Function Editor. This document might contain the following code:

```
<h1>Here is some sample HTML.</h1>
<br />
This HTML will be embedded as a page component.
<br />
```

Using the Web Function Editor, you must now register this page as a Page component. The Page component could be named SAMPLE and defined for output mode. The SAMPLE component is linked to the SAMPLEOUT page.



In your HTML page, you would now use the following statement to include the component:

```
<RDML COMPONENT="SAMPLE" MODE="O">
```

When the tag is processed, the SAMPLE component definition will be read and the content of the SAMPLEOUT page will be embedded.

For the input mode component, you might create a file named SAMPLEIN. The file would be registered to a Page component also named SAMPLE, but the component would be defined for input mode.

Add Page C	omponent	×
Component:	SAMPLE	
Description:	Sample HTML Page	
Page:	SAMPLEIN	
Mode © Inp © Ou © No		
OK	Cancel	

## **9.3 Generated Web Components**

Unlike the 9.2 Manually Defined Web Components, generated Web components have their definitions automatically created by LANSA for the Web. For example, LANSA for the Web can automatically generate all of the HTML code required for a visual drop down Web component.

LANSA for the Web has the following types of generated Web components for use with Web functions:

### 9.3.1 Visual Web Component

### 9.3.4 File Web Component

The main difference between a Visual Web component and a File Web component is that a Visual Web component is created as an HTML page in the LANSA internal database, while the File Web component is created in a library or directory on the Application/Data Server.

You may also wish to review the following:

9.3.2 Creating Visual Web Components

9.3.3 Using Triggers to Generate Visual Web Components.

It is important to note that the Visual and File Web components can also be created manually.



WEB008 - Web Components

### 9.3.1 Visual Web Component

LANSA for the Web can automatically generate four types of Visual Web components for use with Web functions:

- Check Box
- Drop Down
- List Box
- Radio Buttons.

Drop downs, list boxes and radio buttons can be associated with a file to define their values, whereas a check box requires a defined value and description. All of these components are generated as Input mode.

For example, the Department file (DEPTAB) contains a field called Department Code. You can use a Visual Web component to have this field displayed as a drop down list. The values in the drop down list can be based on the contents of the Department file. LANSA for the Web will automatically create the HTML page.

Generate Visual Component
Component: DEPTMENT
Extension: HTM
Sec. Ext.:
Visual Type: Drop down
Repository Data File: DEPTAB
Field for Value: DEPTMENT
Field for Description: DEPTDESC
Use Library List           Include blank entry
Static Data
Value:
Description:
OK Cancel

The HTML page generated might appear like this: <RDML CHECKVALUE="YES">

```
<select size="1" name="DEPTMENT">
<option value="ADM">ADMINISTRATION DPT</option>
<option value="AUD">INTERNAL AUDITING</option>
<option value="FLT">FLEET ADMINISTRATION</option>
<option value="GAC">GROUP ACCOUNTS DEP</option>
<option value="INF">INFORMATION SERVICES</option>
<option value="LEG">LEGAL DEPARTMENT</option>
<option value="MKT">MARKETING DEPARTMENT</option>
<option value="R&D">RESEARCH & DEVELOP</option>
<option value="TRVL">TRAVEL DEPARTMENT</option>
```

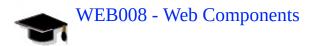
```
</RDML>
```

**Note:** Make sure that the values entered into the Repository Data section are correct. That is, the file you specify exists and contains the fields you specify.

It is important to note that the HTML is based on the contents of the file (LANSA table) at the time the Visual Web component is built. If a new Department was added, you must rebuild the Web component. Also, refer to 9.3.3 Using Triggers to Generate Visual Web Components.

Visual Web components are really just a special type of Page Web component. If you manually define a Visual Web component, you are simply defining a page associated with the component. It is almost identical to defining a Page Web component with the exception that Page Web components are able to support the Not Applicable mode. Visual Web components can only be defined as Input Mode or Output mode.

For details about creating a Visual Web Component, refer to 9.3.2 Creating Visual Web Components.



## 9.3.2 Creating Visual Web Components

There are three methods for creating/maintaining Visual Web Components for use with Web functions:

- Automatically (Input mode only)
- Manually (Input & Output mode)
- Using LANSA Triggers

Automatically created Visual components are built using the *Components*, *Generate Component - Visual* option from the Web Function Editor. You can create Check Box, Drop Down, List Box and Radio Buttons. These components are all defined as Input mode. Refer to Generate Visual Component.

Visual Web components can be manually defined. You will manually create Visual Web components when you need output components. Refer to Visual Component. **Reminder:** You can use a Page Web component instead of manually defining a Visual Web component.

LANSA Repository triggers can also be used with your application database so that the contents of a file are used to update a Visual Web component. Refer to 9.3.3 Using Triggers to Generate Visual Web Components.

## 9.3.3 Using Triggers to Generate Visual Web Components

LANSA for the Web provides you with a program, W3@P2600, which can be used to build input Visual Web components from data contained in a physical file on the Data/Application Server. There is also a LANSA built-in function (BIF) called WEB\_BUILD\_COMPONENT which can be used to call W3@P2600. The W3@P2600 program and WEB\_BUILD\_COMPONENT BIF can be used by LANSA functions to build components. By writing a LANSA function to build or rebuild a component, you can automate the maintenance of Web components.

It is recommended that you use the BIF rather than calling W3@P2600.

As an example, you will use the DEPTAB table from the Personnel Demonstration system in LANSA. The DEPTMENT field has a Visual Web component which is a drop down.

The parameters of the W3@P2600 program are:

- Name of the partition
- Name of the Web component
- Type of visualization
- Name of the physical file on the iSeries
- Name of the field in the physical file whose value will be used with the VALUE keyword in the generated HTML
- Name of the field in the physical file whose value will be displayed as the description in the Visual Web component

This program can be called from a trigger function on the DEPTAB file. This trigger is set up to execute when the contents of the file changes. For example, when the contents of the DEPTAB table changes, the trigger will execute and call W3@P2600 to rebuild the DEPTMENT Visual Web component. Using this approach, the Web application will always have the most recent information without waiting for developers to update components.

**Note:** The W3@P2600 program only supports physical files. This means that you cannot create a Web component using a logical file.

## 9.3.4 File Web Component

The File Web component allows you to use an external file to store HTML. The file will be treated as a data stream file. The contents of the file will be sent to the browser by LANSA, without interpreting any of the data streams contained in the file. This component allows you to circumvent the 256-character limitation in LANSA. This component is mainly used with output fields.

On the iSeries, the file is a standard physical file. The physical file will be created when the file component is created. The library must be defined as part of the file name.

On Windows, the file is a standard text file. The text file will be created when the file component is created. The directory must be defined as part of the file name.

A LANSA Object Access Module is not required to use the file, i.e. the file is not identified to the LANSA Repository.

For example, you might have a large text document which needs to be displayed as part of a Web page. This document is maintained as a standard PC word processing file. Using tools like Word, you can save this document as an HTML file. When the File Web component is embedded into your page, this information will be sent to the browser.

Using the File Web component means that you do not have to create and edit large documents within the Web Function Editor. (Remember, all files created using the Web Function Editor are stored internally in the LANSA database.) Also, because you are using a standard file object, it is also possible to execute programs which read and write from this file before it is used in your Web page.

File Web components can be created manually and they can be automatically generated by LANSA for the Web. If the File Web component is created manually, then the File must be manually defined. LANSA for the Web will create the file if it is automatically generated.

For details about manually creating a File Web component, refer to File Component.

For details about automatically creating a File Web Component, refer to Generate File Component (HTML mode).

## **iSeries** Files

On the iSeries, your File Web components may use library lists. Library lists can be very powerful. For example, if your application requires you to display

different data for the same field, depending on the user profile used to run the application, then a library list can be used with the user profile and File Web component.

A different file can be created for each user profile. Each file contains the HTML to display the data. These files would be installed to the appropriate library. The library list attached to the user profile would then be used to locate the correct file.

When the application is executed, LANSA for the Web will use the File Web component. However, it will use the library list attached to the user profile to locate the correct file, since no library is attached to the file definition in the Web component registry.

### **10. Function Editor**

The Web Function Editor is used to enhance your Web Function applications. If you are not familiar with the Web Function Editor, you should review:

10.1 Introduction to Web Function Editor

The Web Function Editor includes the following menu categories:

10.2 File Menu
10.3 Edit Menu
10.4 View Menu
10.5 Tags Menu (HTML mode)
10.6 Components Menu
10.7 Options Menu
10.8 Tools Menu

### **10.1 Introduction to Web Function Editor**

In order to execute the Web Function Editor, you should review the following topics:

- 10.1.1 What is the Web Function Editor?
- 10.1.2 Connecting to the Data/Application Server
- 10.1.3 Starting the Web Function Editor
- 10.1.4 Web Function Editor's Main Window

## 10.1.1 What is the Web Function Editor?

The Web Function Editor provides you with full text editing capabilities to create and modify, the HTML/XML pages generated for your Web function application. You manage your application's Components and Graphic Variables using the Web Function Editor. The Editor also allows you to create and maintain documents supporting XML transformations (for example XSL style-sheets).

To accommodate specific HTML or XML editing tasks, you will need to specify whether you want to run the Web Function Editor in XML or HTML mode.

If you have more generic tasks, you can use the BASIC mode. BASIC mode provides the functionality valid for all modes and is a sub-set of the other modes (i.e. XML or HTML).

Throughout this section of the document, mode-specific tasks are labeled as such. Where there is no label (e.g. XML only) the task is valid for both modes. Features not available with the current mode are grayed out on the Editor's dialog boxes.

The Web Function Editor does not provide a facility for you to edit the generated HTML/XML graphically as the LANSA generated pages include LANSA Web components and LANSA tags. The effect of some LANSA tags will not be shown until the LANSA function is run. In addition, if your partition is multilingual, you will not be able see the descriptions of the fields, since these are inserted when the function is executed.

 $\uparrow$  10.1 Introduction to Web Function Editor

### **10.1.2** Connecting to the Data/Application Server

To use the Web Function Editor, LANSA Open must be installed. If, during the installation, LANSA Open is not detected, the LANSA for the Web Function install will install, by default, the components required for TCP/IP connectivity between your workstation and the host.

## **User Profiles**

When you start the Web Function Editor you will need to connect to the Data/Application Server using a valid user profile.

For the AS/400, the user profile specified must be properly configured to use the LANSA system.

 $\uparrow$  10.1 Introduction to Web Function Editor

## **10.1.3 Starting the Web Function Editor**

When you install the Web Function Editor, a shortcut icon is created on your desktop. When you double click on the shortcut icon, the System Defaults dialog is displayed.

System Default	S		×
Profiles:	MYSYSTEM		•
Profile Details LANSA System:	MySystem		Save <u>A</u> s
Host Type:	IBM i 🗨		<u>S</u> ave
Partition:	DEM		<u>D</u> elete
Language:	ENG		<u>R</u> eset
Userid:	User		
Password:	*****		
Mode:	HTML -		
	Save Password Auto-connect on startup		Advanced
ОК	Cancel		

Enter the connection parameters for the required system.

If you have previously saved the connection details as a Profile, select the Profile that you require.

For more details about the parameters, go to the 10.2.13 Connect command on the File menu.

#### Mode

The modes will be either:

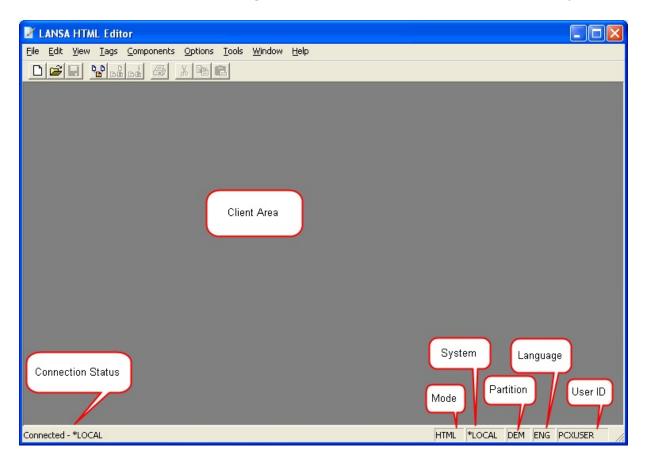
- XML
- HTML or
- BASIC. Basic is used for more generic tasks. This mode supports functionality valid for all modes and is a sub-set of the other modes (e.g. XML or HTML).

Once you are connected successfully to the host, the 10.1.4 Web Function Editor's Main Window is displayed.

## 10.1 Introduction to Web Function Editor

## **10.1.4 Web Function Editor's Main Window**

You use this main window to gain access to all the Editor's functionality.



The status bar at the bottom of the window displays your current connection status, the mode, system, partition, language and User Id.

To change any of the connection details, or to reconnect to a different system, choose the Connect command from the File menu. You will be asked to confirm disconnecting from the current system. If you answer Yes, the System Defaults dialog box will be opened and you can reconnect to the system.

↑ 10.1 Introduction to Web Function Editor

## 10.2 File Menu

The File Menu contains the following options:

10.2.1 New

10.2.2 Open . . .(XML/HTML mode)

10.2.3 Open (BASIC mode)

10.2.4 Close

10.2.5 Save

10.2.6 Save As

10.2.7 Save As (mode BASIC)

10.2.8 Save To Local (mode BASIC only)

10.2.9 Load From Local (mode BASIC only)10.2.10 Compare

10.2.10 Compare

10.2.11 Page Setup

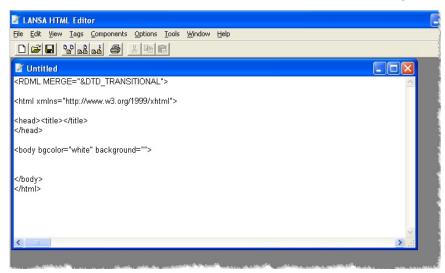
10.2.12 Print

10.2.13 Connect

### 10.2.1 New

Use New command to define a new HTML/XML page.

If you are running in HTML mode, when you select this command, the Editor pre-fills the client area with a default set of HTML/XHTML tags.



Once you have defined the new HTML/XML page, it can be saved using either the Save or Save As command from the File menu.

If you intend to use this newly created page as a LANSA Web component, you will need to remove the first two lines (RDML MERGE..., <html xmlns....) and register it as a Web component as described in Add a new Component. It can then be used in conjunction with the <RDML COMPONENT> tag.

# 10.2.2 Open . . .(XML/HTML mode)

If you are using the Open command in BASIC mode, then go to 10.2.3 Open (BASIC mode).

Use the Open command to select existing HTML/XML pages stored in the LANSA Repository. When you choose this command, you are presented with a dialog box listing all the HTML or XML pages (depending on the mode) currently stored in the partition.

Open HTM	L Page	
PSLSYS PSLSYS PSLSYS PSLSYS PSLSYS PSLSYS PSLSYS PSLSYS	INQUIRE001 INQUIRE002 PHONE 001 SALARY 001 SEARCH 001 SEARCH 002 SEARCH 003 SNAME 001 WINDOW 001	Process PSLSYS Func Process PSLSYS Func
	WINDOW 002	Process PSLSYS Func
Name: Language:	SEARCH 003"	Version: 0
Description: 002" "Process PSLSYS Function SEARCH 003"		
OK <u>D</u> elete Cancel		

To edit the HTML/XML pages, select one or more entries from the list of pages and press *OK*. To identify the HTML page for a specific function, refer to Identifying Generated Pages.

If you enter the first character of the Page's name, the list will be positioned at the beginning of names starting with that character.

To delete HTML/XML pages from the LANSA Repository, select one or more entries from list of pages and press the *Delete* button. You will be asked to confirm your deletion instruction.

### Version

The version number of the current HTML/XML page is 0. The previous copy of the page will be version number 1, and so on. For more details, refer to Versioning of Pages.

### Language

If you are working with a multilingual partition, you can choose the language of the page from the Language drop down list.

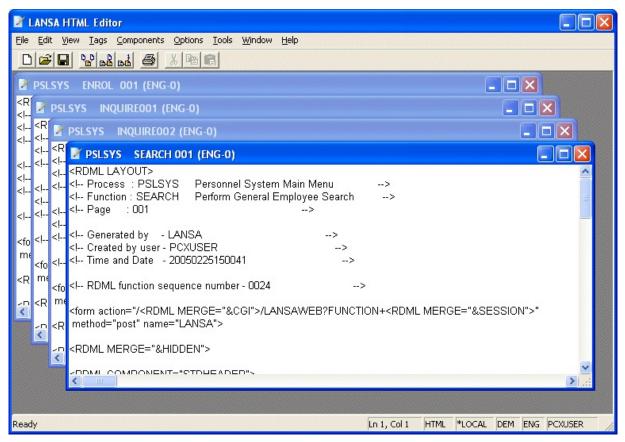
### Second. Extension (XML Mode)

In HTML mode, this field will be hidden.

Secondary extension specifies the sub-extension (also called the XML Application) to be used to identify the Component. This value enables you to simplify the search for XML documents.

# **Opened Documents**

When you select one or more pages to edit and press *OK*, the HTML/XML pages will be displayed in the client area of the Editor. The pages will be downloaded from the Application/Data Server. The download time will depend upon the size of the documents and the speed of your connection. Once the pages have been downloaded, you can then begin to edit the page.



Each page opened is displayed in its own window. You can toggle between the pages you are editing by selecting the Windows command from the menu bar and choosing Tile Horizontally or Tile Vertically.

The HTML/XML page you are editing is shown in the Title Bar.

Once you have finished editing, the pages are kept in memory of the local PC until they are saved to the Data/Application Server.

10.2 File Menu

## 10.2.3 Open (BASIC mode)

If you are using the Open command in HTML or XML mode, then go to Open . . .(XML/HTML mode).

The Open command displays the Open Page dialog box. This dialog box is divided into Selection, List and Details areas.

Open Page	
Extension: HTM	Second. Extension:
DEBUG_SCRIPT	Script function for debug
DEFAULT_CONTENT	Default Process Content 📃
DEFAULT_FRAMESET	Default FRAMESET
DEFAULT_HIDDEN	Default hidden fields
DEFAULT_INDEX	Default Process Menu page
DEFAULT_LAYOUT	Default Layout
DEFAULT_SCRIPT	default script
DEFAULT_STYLE	Default Cascading Style Sheel
DEFAULT TRACE	Default Trace
Extension: HTM	Second. Extension:
Name:	Version: 0
Language: ENG 💌	
Description:	
OK	Delete

#### **Selection Area**

This area, in the upper part of the dialog box allows you to enter the search criteria for the documents to be displayed in the **List**. The list will be populated once you press the List button.

#### Extension

Select the file extension to be used to build the list of documents. Your selection will be:

- HTM for HTML documents
- XML for LANSA XML documents

#### Second. Extension

Enter the Secondary Extension (called the XML Application in the Administrator) to be used when retrieving the list of documents. If you leave this field blank, all the documents matching the Extension will be listed.

#### List

Press the *List* button to retrieve the documents matching the settings you have entered in Extension and Second. Extension. The documents retrieved will be displayed in the list. Depending on your selections, it may take some time before the list is downloaded and displayed.

#### **Details Area**

This area displays the details for a document(s) that is highlighted in the list.

#### Extension

Displays the file extension for the selected document(s).

#### Second. Extension

Displays the Secondary Extensions (or XML Applications) for the selected documents.

#### Version

The version number of the current page is 0. The previous copy of the page will be version number 1, and so on.

#### Language

If you are working with a multilingual partition, you can choose the language of the page from the Language drop down list.

#### Description

The description of the document that is selected.

### 10.2.4 Close

The Close command closes the current page in the Editor.

If you made changes to a page and you have not saved the changes, you will be prompted to save the changes before the Editor closes the page.

10.2 File Menu

### 10.2.5 Save

The Save command will save the current Page to the Application/Data Server. The time to save the file will depend upon the size of the document and the connection to the server. The cursor will change to an hour glass and the save status is displayed on the status bar. Do not execute the application until the document save has completed.

If the current page is untitled, you will be prompted to provide a name and description for the page (please refer to the following sections: 10.2.6 Save As for XML or HTML pages, or 10.2.7 Save As (mode BASIC) for all other pages.

You can only save changes if you are editing the version 0 page. You must use the Save As option if you are editing an archived page (version 1 through 10).

If you have configured the Editor options to Enable archive functionality when saving, you will be asked if you wish to archive previous versions of the file before it is saved. For more details, refer to the Miscellaneous option of the 10.7.1 Configure command.

For more details about versions, refer to Versioning of Pages in LANSA Generated HTML Pages.

If you lose your connection to the Application/Data Server while saving or before saving, you should copy the document contents to another file on your PC. Simply open another editor (such as NOTEPAD.EXE) and cut and paste the text to this editor. Once you have reconnected to the server, open the file you were saving. Cut and paste the text back into the Web Function Editor and try saving again.

### 10.2.6 Save As

The Save As command allows you to save an existing HTML/XML page with a different page name.

If you have opened an archived page (version 1 through 10), the changes can only be saved as version 0. For example, if you open version 3 of a page, you cannot save this page as version 3. It must be saved as version 0. This rule ensures that your previous versions are not corrupted. For more details about versions, refer to Versioning of Pages.

Also refer to the 10.2.5 Save command for more details about saving files back to the Application/Data Server.

## 10.2.7 Save As (mode BASIC)

Selecting the Save As command when you are in BASIC mode will open the enhanced Save Page dialog box. This dialog allows you to list the known pages for the given Extension and Second. Extension or to directly specify the parameters for the new page in the Details area.

Save Page		
Extension: HTM 💌 Sec	cond. Extension:	List
PSLSYS ENROL 001	Process PSLSYS	Function
PSLSYS INQUIRE001	Process PSLSYS	Function I
PSLSYS INQUIRE002	Process PSLSYS	Function I
PSLSYS PHONE 001	Process PSLSYS	Function I
PSLSYS SALARY 001	Process PSLSYS	Function !
PSLSYS SEARCH 001	Process PSLSYS	Function !
PSLSYS SEARCH 002	Process PSLSYS	Function !
PSLSYS SEARCH 003	Process PSLSYS	Function !
PSLSYS SNAME 001	Process PSI SYS	Function !
<		>
Details Extension: HTM 💌	Second. Extension:	
Name: PSLSYS ENROI		
Language: ENG 💌		
Description: Process PSLSYS	Function ENROL 001	
ОК	Canc	el

The file extension and secondary extension in the Selection area are those that are current for the document. If you want a different file Extension and Second Extension (i.e. secondary extension) saved with the document, you must enter them in the Details area.

## 10.2.8 Save To Local (mode BASIC only)

This command allows you to save the current page to a file on a local drive. Using a standard File Save As dialog, you specify the name of the file you want to save the current page to. Any existing files of the same name will be overwritten.

## **10.2.9 Load From Local (mode BASIC only)**

This command displays a standard windows File Open file dialog allowing you to select a file. After pressing Open on the standard File Open dialog you will be asked to confirm that you want to overwrite the contents of the current page with the contents of the selected file. After confirming this, the content of the current page will be replaced by the content of the file selected and the current page will be marked as modified.

To replace the page on the host with the page loaded from Local, you can either Save it with the same name or select the Save As command to save it with a different name.

10.2 File Menu

## 10.2.10 Compare

To compare HTML, XML or Basic pages, select Compare from the File menu and choose one of these commands, as appropriate:

- New Comparison use when you do not have any pages open.
- Compare With when you want to compare with a page that is already open.
- Compare With Version when you want to compare with another version of a page you already have open.
- When you have finished with the Compare select Close.

### Version saved

Before you can compare different versions of your pages, you must first specify how many versions of the pages that you want to save. To specify the number of versions you want to save, set the number of copies on the Backup page of the Data/Application Server Configuration in the LANSA for the Web Administrator.

### Save old before over-writing

If the backup feature is enabled in the Data/Application Server Configuration in the LANSA for the Web Administrator, then, when LANSA generates the HTML/XML for a particular Web function, it will save the current page before generating a new page for the function being compiled.

### Save with versioning

If previous versions are being saved (specified in the Backup page of the Data/Application Server Configuration in the LANSA for the Web Administrator) any changes made to an HTML/XML/BASIC page will be saved to the version number opened.

### 10.2 File Menu

## **New Comparison**

You use the New Comparison command when you do not have any HTML, XML or BASIC pages open.

Compare HTML Page 1			
PSLSYS EMPLIST001	Process PSLSYS Func		
PSLSYS ENROL 001	Process PSLSYS Fund		
PSLSYS ENROL2 001	Process PSLSYS Func		
PSLSYS INQUIRE001	Process PSLSYS Func		
PSLSYS INQUIRE002	Process PSLSYS Func		
PSLSYS PHONE 001	Process PSLSYS Func		
PSLSYS SALARY 001	Process PSLSYS Func		
PSLSYS SEARCH 001	Process PSLSYS Func		
PSLSYS SEARCH 002	Process PSLSYS Func		
PSLSYS SEARCH 003	Process PSLSYS Func		
DOLOVO CNAME 001	Deserve Del eve Euro		
Name: PSLSYS ENROI	Version: 0		
Language: ENG 💌			
Description: Process PSLSYS Function ENROL 001			
ОК	Cancel		

Select the first page that you want to compare in the Compare dialog box. When you press *OK*, the Compare dialog box will open again for you to select the second document to compare with the one you have just selected.

When the pages are displayed, the differences between the two pages you have selected are highlighted, using the colors you have chosen in the View page of the Editor's Configuration options.

LANSA HTML Editor - [Comparison of PSLSYS ENROL 001 (ENG-0):PSLSYS E	NROL 2 00	1 (ENG-	0)]		
∦ Eile Edit View Iags ⊆omponents Options Iools Window Help				_ 8 >	<
Comparison of PSLSYS ENROL 001 (ENG-0) PSLSYS ENROL2 001 (ENG-0)				<u>^</u>	
<rdml layout=""> <i :="" main="" menu="" personnel="" process="" pslsys="" system=""></i></rdml>					
<i :="" main="" menu="" personnel="" process="" pslsys="" system=""> +&gt;<i :="" a="" employee="" enrol="" function="" new=""></i></i>					
-< Function : ENROL2 Enrol a New Employee					
Page : 001					
<i -="" by="" generated="" lansa=""></i>					
<pre><!-- Created by user - PCXUSER--> +&gt;<!-- Time and Date - 20050225150030--></pre>					
-< Time and Date - 20050225150031					
RDML function sequence number - 0017					
<form &cgi"="" action="/&lt;RDML MERGE=">/LANSAWEB?FUNCTION+<rdml merge<br="">method="post" name="LANSA"&gt;</rdml></form>	E="&SESS	SION">"			
<rdml merge="&amp;HIDDEN"></rdml>					
<rdml component="STDHEADER"></rdml>					
<center><h1><rdml merge="&amp;FUNCTION"></rdml></h1></center>					
 				~	-
<u>&lt;</u>				>	
Ready	HTML *I	OCAL I	DEM ENG	PCXUSER	11.

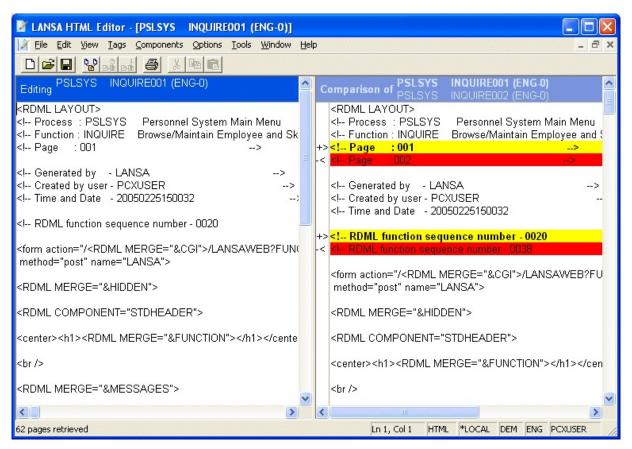
↑ 10.2.10 Compare

## **Compare With**

Select *Compare With* if you already have a page open and you would like to compare that page with an archived page. When you select the *Compare With* command, the Open HTML/XML/Basic Page dialog box (depending on the current mode) is opened so that you can choose the page you want to compare with your current page.

Open HTML Page	×		
PSLSYS ENROL2 001 PSLSYS INQUIRE001	Process PSLSYS Func		
PSLSYS INQUIRE002	Process PSLSYS Fund		
PSLSYS PHONE 001 PSLSYS SALARY 001 PSLSYS SEARCH 001 PSLSYS SEARCH 002 PSLSYS SEARCH 003 PSLSYS SNAME 001	Process PSLSYS Func Process PSLSYS Func Process PSLSYS Func Process PSLSYS Func Process PSLSYS Func Process PSLSYS Func		
PSLSYS WINDOW 001	Process PSLSYS Func		
Name: PSLSYS INQUIF	Version: 0		
Language: ENG  Description: Process PSLSYS Function INQUIRE 002			
Description: Process PSLSYS Fur	ICTION INQUINE 002		
ОК	Cancel		

The pages are displayed in the orientation that you have specified in the View page of the Editor's Configuration Options.



The vertical display feature allows you to locate the differences between the two pages quickly as well as allowing you to incorporate any changes into the page you are editing.

To close the comparison window, select *Close* from this window's File menu.

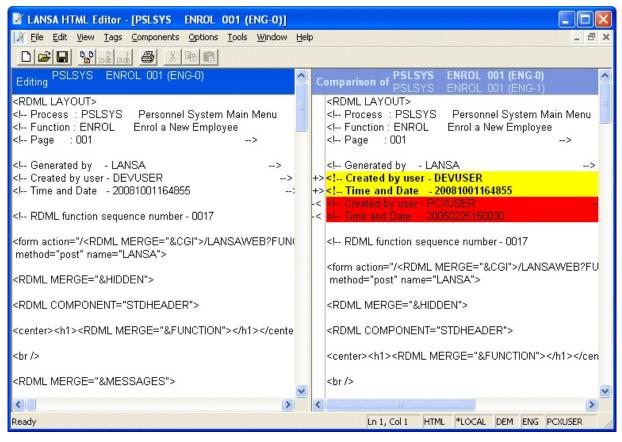
10.2.10 Compare

### **Compare With Version**

Choose the *Compare With Version* when you are editing an HTML/XML or BASIC page and you would like to compare the page with an archived version of the same page. The HTML Page Version dialog box is displayed.

HTML Page Version	
Page Vei	rsion: 🚺 📩
OK	Cancel

Enter the Page Version that you wish to open and press *OK*. The page you have selected will be displayed side by side (or whatever orientation you have chosen for comparisons when selecting your required options in the View configuring the Editor) with the differences highlighted.



10.2.10 Compare

# Close

To close the comparison window, choose *Close* from this window's File menu. If you were editing an HTML/XML or BASIC page, you will be returned to that window.

10.2.10 Compare

## 10.2.11 Page Setup

When you choose the *Page Setup* command from the File menu, the standard Windows Page Layout dialog box is displayed so that you can customize the page layout.

10.2 File Menu

### 10.2.12 Print

This command opens a standard Windows Print dialog box and prints the HTML/XML or BASIC page currently open. You can choose to print the entire page or a highlighted selection.

# 10.2.13 Connect

Use the *Connect* command on the File menu to connect to a different host or to change from the current Mode (HTML, XML or BASIC) to a different Mode.

When you have specified the details of the system to which you want to connect in the Profile Details area, you can save the details as a Profile. Once you have assigned a Profile name, you simply select a profile rather than enter the profile details each time you wish to reconnect.

System Default	s <sup>elen orden en de la </sup>	
Profiles:	MYSYSTEM	<b>_</b>
Profile Details LANSA System:	MYSYSTEM	Save <u>A</u> s
Host Type:	iSeries 🔻	<u>Save</u>
Partition:	рем	<u>D</u> elete
Language:	ENG	<u>R</u> eset
Userid:	User	
Password:	******	
Mode:	HTML - Save Password Auto-connect on startup	<u>A</u> dvanced
ОК	Cancel	

Although this dialog is similar to the System Defaults used by the LANSA for the Web Administrator, there is no relationship with between the Profiles recorded using Administrator's System Defaults and the Profiles recorded using the Editor's System Defaults.

#### Mode

Enter the execution mode for the Editor. Possible modes are HTML, XML or BASIC. Depending on the mode you choose, the mode-specific options will be enabled/disabled once you are connected to a host system. The BASIC mode is a more generic mode, allowing you to perform tasks common to all type of pages (HTML, XML, XSL etc.).

If you select a Host Type of Other, then only Mode HTML and XML will be available for selection.

#### **Save Password**

You can save the password for future use by selecting this option.

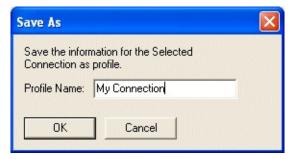
## Auto connect on startup

If you select this option, and save the settings as a Profile, you will be connected automatically to this system when you next start the editor.

Save As Advanced ↑ 10.2 File Menu

## Save As

If you want to reuse the connection parameters, then you can save them as a Profile. Select the Save As... option and enter the Profile Name in the Save As dialog box. Press *OK* to return to the System Defaults dialog.



#### Save

Press *Save* to save the current entries with the Profile Name you have specified. If you have changed setting, remember to Save the changes before you press the *OK* button to start the connection process.

#### Delete

To delete a Profile, select the profile to be deleted from the Profile drop down list then press the *Delete* button. You will be asked to confirm the delete request.

You are not able to delete the <Default> profile.

#### Reset

Press *Reset* to return all the parameters for the Profile Details and the related Advanced settings to their default values.

↑ 10.2.13 Connect

## Advanced

When you press the *Advanced* button, the Advanced System Options dialog box is displayed. This allows you to customize replacement characters for the '@' and '#' symbols. You will only need to customize these characters if you are running a non-English system.

Advanced System Options	X
Character to replace '@' with: à	
Character to replace '#' with: #	
OK Cancel	

## 10.2.13 Connect

Test Case:

### **Test Plan Writer: Torkel CronholmDate Written:**

#### LANSA Version: 11.4Date Last Tested:

**Purpose:** Long userid/password: Use existing profiles, create new profiles, modify profiles, delete profiles. Need to test with an earlier release of Web Function Editor as well to ensure backwards compatibility within the constraints defined in the test case. Need to trick the Web Function Editor into believing it is using data from a never version of the Web Function Editor.

#### **Comments:**

Tester:	_Date Commenced:	Date
Completed:		

Test	Expected Results

1	Use the 11.3 version of the program and fill in Profile Details.	The entries are saved.
	Use a combination of entries for LANSA System, Host Type, Partition, Language, Userid, Password, Mode, Save Password and Auto-connect on startup as well as settings under the Advanced area. Save each entry with a new name using "Save As"	Open the 1 version of program ai ensure that profiles are present and that the dat correct.
		Open the 1 version of program ai ensure that profiles are present and that the dat correct.
2	Use the 11.4 version of the program and fill in Profile Details.	The entries are saved.
	<ul> <li>Use a combination of entries for LANSA System, Host Type, Partition, Language, Userid, Password, Mode, Save Password and Auto-connect on startup as well as settings under the Advanced area.</li> <li>Save each entry with a new name using "Save As"</li> <li>Only use Userids and Passwords that are NOT longer than 10 characters.</li> </ul>	Open the 1 version of program ai ensure that profiles are present and that the dat correct.
		Open the 1 version of program ai ensure that profiles are

		present and that the dat correct.
3	Use the 11.4 version of the program and fill in Profile Details.	The entries are saved.
	Change an entry to have Userid to be longer than 10 characters. Save the profile using "Save" Change an entry to have Password to be longer than 10 characters. Save the profile using "Save" Change an entry to have Userid and Password to be longer than 10 characters. Save the profile using "Save"	Open the 1 version of program ai ensure that profiles are present and that the Us and/or Password a blank. Open the 1 version of program ai ensure that profiles are present and that the dat
4	Use the 11.3 version of the program and fill in Profile	correct.
	Details. Fill in Userid and Password for those entries done above and save each entry using "Save"	are saved. Open the 1 version of program ai ensure that profiles are present and that the dat

		correct.
		Open the 1 version of program an ensure that profiles are present and that the dan correct.
5	Use the 11.4 version of the program and press reset for a Profile.	The entries should be blank exce for Host Ty which shou be "IBM i' Mode whic should be "HTML". "Advanced the @ shou be "@" and the # shoul be "#"
6	Use the 11.4 version of the program. Connect to IBM i with a combination of correct short and long Userids and Passwords. Correct here means that the Userid need to exist exist on IBM i and passwords need to be correct. Use all uppercase, all lowercase and a combination of uppercase/lowercase for Password.	Connectioı IBM i shoı succeed
7	Use the 11.4 version of the program. Connect to IBM i with a combination of incorrect short and long Userids and Passwords. Incorrect here means that the Userid must not	Connection IBM i shou not succee

	exist on IBM i and/or passwords are incorrect.	Error mess with return code 0x6 is displayed.
8	Use the 11.4 version of the program. Connect to Windows (Other) with a combination of correct short and long Userids and Passwords. Correct here means that the Userid need to exist exist on Windows and passwords need to be correct. Use all uppercase, all lowercase and a combination of uppercase/lowercase for Password.	. Connection to Window should succeed.
9	Use the 11.4 version of the program. Connect to Windows (Other) with a combination of incorrect short and long Userids and Passwords. Incorrect here means that the Userid must not exist on Windows and/or passwords are incorrect.	Connection Windows should not succeed. E message w return code 0x6 is displayed.
10	Use the 11.3 version of the program and delete a few of the created profiles.	The entries are deleted Open the 1 version of program ai ensure that profiles are not present Open the 1 version of program ai ensure that profiles are

		not present
11	Use the 11.4 version of the program and delete a few of the created profiles.	The entries are deleted
		Open the 1 version of program ai ensure that profiles are not present
		Open the 1 version of program ai ensure that profiles are not present
12	We need to ensure that future backwards compatibility is working, ie, that the program works with the next generation Userid /Passwords.	Open the 1 version of program. <i>F</i>
	11.3 level data has no version number, hence the lack of a version number is interpreted as version 0. 11.4 level data is version 1. The next generation data (for the future) will be any number larger then 1. If 11.3 finds data with userid/password longer then 10 characters it will leave userid/password fields empty (as it does for version 1). If 11.4 finds data of a later version, ie, profiles data's version	profile data you made version number 2 v show its details disabled.
	number is $> 1$ , it will disable the entire connection screen (except for where to chose the profile of course).	Open the 1 version of
	To test this, you need to manually edit the profiles data version.	program. <i>F</i> profile you made to
	This is how to do it: Open registry (using regedit). Go to HKEY_CURRENT_USER\Software\LANSA\LANSAWEB.	version number 1 (

	Under this key look for a PROF# key (# being a number) and add and set or change LW3_Version to 2 (for testing 11.3 and 11.4) or 1 (for testing 11.3). IMPORTANT: REMEMBER TO SET THE LW3_Version VERSION BACK AFTER YOU FINISHED WITH THIS TEST	will show details enabled BU the UserId and/or Password v be displaye as blank if they are longer than characters.
13		

Test Case:

#### Test Plan Writer: Torkel CronholmDate Written:

#### LANSA Version: 11.4Date Last Tested:

**Purpose:** Connecting to backend where the nominated OS type is not the one we actually are connecting to

**Comments:** In all tests ensure that all connection details expect for Host Type are correct. The ONLY thing wrong should be the Host Type.

 Tester:
 \_\_\_\_\_\_Date Commenced:
 \_\_\_\_\_Date

Completed:
 \_\_\_\_\_\_

	Test	Expected Results	Pass/ Fail
1	In Profiles Details specify a	Fails with message: "Connection	

	LANSA System for Windows, but select IBM i as Host Type. Connect.	failed. Connection defined as IBM i but attempt was made to connect to Windows''	
2	In Profiles Details specify a LANSA System for IBM i, but select Other as Host Type. Connect.	Fails with message: "Connection failed. Connection defined as Other but attempt was made to connect to IBM i"	
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		

27		
28		
29		
30		

## 10.3 Edit Menu

The Edit menu commands are provided for you to edit your HTML/XML. The Edit menu's commands allow you to:

Undo Cut Copy Paste 10.3.1 Paste HTML (HTML only) Select All 10.3.2 Clear All Find Replace

Except for the two LANSA for Web specific commands, these editing commands work in the same way as the standard Microsoft Windows commands.

# 10.3.1 Paste HTML (HTML only)

Paste HTML pastes text from the Clipboard into the active window in HTML Format. This means that the text to be pasted includes the formatting HTML tags (unlike the Paste command that only copies the text - not the HTML tags).

10.3 Edit Menu

## 10.3.2 Clear All

This command deletes all the HTML/XML lines from the document that is currently displayed. You are asked to confirm your deletion before the lines are actually deleted.

10.3 Edit Menu

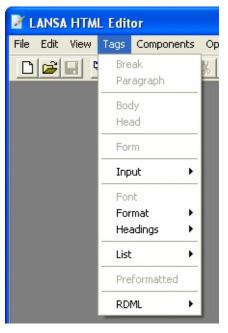
## 10.4 View Menu

The View menu allows you to control the display of the Toolbar at the top of the main window and the Status Bar at the bottom of the main window.

The display options are a toggle. If you select the Toolbar, it will be displayed. If you select it again, it will not be displayed.

## 10.5 Tags Menu (HTML mode)

Use the Tags menu to insert HTML tags into your HTML page. The tags will be inserted at the current cursor position.

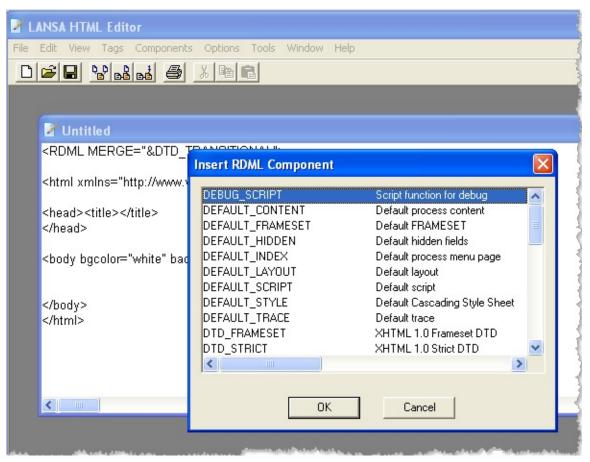


You will find the most commonly used HTML and LANSA tags in the Tags sub menus.

10.5.1 Insert RDML Component

## **10.5.1 Insert RDML Component**

When you choose the Component tags from the RDML sub-menu, the RDML Component dialog box is opened to display the list of Web components currently registered in the partition's Web Component Registry.



If you enter the first character of the Component's name, the list will be positioned at the beginning of the names starting with that character.

Select a Component from the list and press *OK*. The component will be automatically inserted into your HTML page.

10.5 Tags Menu (HTML mode)

## **10.6 Components Menu**

The Components Menu contains the following options:

10.6.1 Maintain Component

10.6.2 Generate Component

10.6.3 Graphic Variables

# **10.6.1 Maintain Component**

You use the Maintain command on the Components menu to maintain your existing Web components or to add new Web components.

If you are running Task Tracking in LANSA, then when you add, change, or delete Components, the Administrator will prompt you for the Task Id required by the Task Tracking level.

When you select *Maintain* from the Components menu, the Components dialog box is displayed with a list of the Web components in the Web Components Registry of the current partition. From this dialog box you can Change or Delete the components that are listed or you can Add new Web components. The dialog boxes used to change or add Web components will vary depending on the type of component you are working with.

Maintain Components				D
Component	Туре	Description	Value	Extension(s)
DEBUG_SCRIPT	Page	Script function for	DEBUG_SCRIPT	НТМ
DEFAULT_CONTENT	Page	Default process c	DEFAULT_CONTENT	HTM
DEFAULT_FRAMESET	Page	Default FRAMESET	Add Insert	НТМ
DEFAULT_HIDDEN	Page	Default hidden fields	Change	HTM
DEFAULT_INDEX	Page	Default process m	Delete Del	HTM
DEFAULT_LAYOUT	Page	Default layout	Duplicate	НТМ
DEFAULT_SCRIPT	Page	Default script	DEFAULT_SCRIPT	НТМ
DEFAULT_STYLE	Page	Default Cascading	DEFAULT_STYLE	НТМ
DEFAULT_TRACE	Page	Default trace	DEFAULT_TRACE	НТМ
DTD FRAMESET	Page	XHTML 1.0 Frame	DTD FRAMESET	нтм 💌
<				
Close		Add	Change Delete	1

## Add

Press the *Add* button to Add a new Component.

## Duplicate

If you select a component and press the right mouse button, a floating menu is displayed. On this menu are these menu items: *Change*, *Delete* and *Duplicate*. If

you select *Duplicate*, the selected component will be used as a template for a new component and its details will be displayed in the Add dialog box for the type of component you have selected.

With this feature you can easily create components without having to enter similar values over and over again.

### Change

Select the component in the list and press the *Change* button. The dialog box that is displayed will depend on the type of component you are changing.

### Delete

To delete a component from the Web Component Registry, select the component to be deleted in the list and press the *Delete* button. You will be asked to confirm the deletion.

Add a new Component Duplicate Banner Component (HTML mode) File Component Page Component Script Component (HTML mode) Text Component Visual Component Web Link Component (HTML mode)

## Add a new Component

After you press the *Add* button on the Maintain Components dialog box, the Add Component dialog box is opened. This is always the first step in defining a new Web component unless you use the **Duplicate** feature.

Add Compo	nent 🛛 🔀
Component:	
Extension:	HTM
Sec. Ext.:	
Туре:	Banner 💌
Continue	. Cancel

#### Component

The new Web component's name. This is the name you will use in conjunction with the <RDML COMPONENT> LANSA tag.

#### Extension

The file extension. This indicates the mode in which you are processing. If you are in BASIC mode, you will need to select the file extension you require.

#### Sec. Ext. (XML mode)

Secondary extension specifies the sub-extension (also called the XML Application) to be used to identify the Component. This value enables you to simplify the search for XML documents.

#### Туре

Select the Type of component from the drop down list of possible Web component types.

#### Continue

To enter the component's details, press the *Continue* button. The details required depends on the Type of component that you are creating. It could be a:

- Banner Component (HTML mode)
- File Component
- Page Component
- Script Component (HTML mode)

- Text Component
- Visual Component
- Web Link Component (HTML mode)
- 10.6.1 Maintain Component

# Duplicate

If you are adding many new components, you can use the Duplicate feature to easily create components without having to enter similar values over and over again.

To use the Duplicate feature, simply select the component on which to base your new component and right click with the mouse to display the pop-up menu. Select *Duplicate* from the pop-up menu. The appropriate Add component dialog box is opened with the entry fields pre-filled with the values of the selected component.

10.6.1 Maintain Component

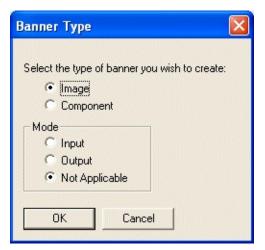
# Banner Component (HTML mode)

Banner Web components allow you to create a banner as a list of images or as a list of Web components. Banner Web components are used when you wish to change the presentation or layout of your page periodically.

When defining the banner Web component as a list of Web components, the Web components can consist of any type of Web components. For example, you could create a Banner Web component that consists of a Visual drop down Web component, check box Web component, radio button Web component. In addition, the Banner Web component can embed another Banner Web component.

This allows you to customize the presentation of your data. A Banner Web component can be embedded into any HTML page in your application.

When you have chosen to create a Banner Web component, the Banner Type dialog box is displayed.



## **Banner Type**

Select the type of banner you wish to create. A banner will consist of either images or Web components. Your selection will be shown in the following dialog box where you specify further details of the banner you have selected.

#### Mode

You can choose to define the Web component as a mode dependent Web component by selecting the appropriate mode. For more details, refer to Web Components and Modes.

10.6.1 Maintain Component

# **Add/Change Banner Component**

When you select the *OK* button, the Add or Change Banner Component dialog box, as appropriate, is displayed. If you are creating a number of Banner Components, you can use the **Duplicate** feature.

Add Banner Component	
Component: WEBSTART Description:	Banner type: Image Display link in new window
Update frequency: Update every visit Update every week Update every n days Update every n months Banner Items	
Set Image Description	Click Count URL
Add	inge
OK Cancel	

The Banner type you have chosen is displayed in the top right corner of the box. You can work with either a set of images or a set of components. You use the Add, Change or Delete to modify the definition of the banner Web component.

#### Description

The description of the Web component in the Web Component Registry.

#### Display link in new window

If this option is selected, it will open the URL / Link on a particular banner image in a new browser window. By Default, the URL / Link on a banner image will open in the current browser window.

## **Update frequency**

Select the appropriate frequency in the list. Additional options will be requested,

depending on the frequency selected.

- **Update every visit** Cycle through each image or component on a per visit basis (i.e. each time the banner component is used it will be updated to the next item).
- **Update every week** Cycle through each image or component on a particular day of the week. For example, you could create a Banner component that changes every Monday.
- **Update every n days** Cycle through each image or component based on number of days. For example, you could create a Banner component that changes every 50 days.
- **Update every n months** Cycle through each image or component on a monthly basis. For example, you could create a banner Component that changes every 3 months on the 10<sup>th</sup> day of that month.

## **Banner Items**

If you are using multiple banner images in a cycle, the images in the cycle are shown in the Banner Items list in the dialog box.

## **Make Current**

In the list of images, the image that is current is indicated by a tick in the Set column.

To make a different image the current image, select the image you wish to make current, right mouse click to bring up the popup window and choose the Make Current command. This image will be the current image the next time the Banner Web component is called.

Press the *Add*... or *Change* button to provide further details.

↑ Banner Component (HTML mode)

# Add/Change Banner Detail

You reach the Add or Change Banner Detail dialog box by selecting the *Add*... or *Change* button on the Add/Change Banner Component dialog box.

Add Banne	r Detail	×
Set: Image: Description: Click Count: URL:	0 Reset Click Count	
ОК	Cancel	

### Set

The number specified in Set controls the sequence in which images are cycled. You should start at 0 and increment by 1 for each image. LANSA for the Web assumes that the images assigned to the Banner Web component are assigned a sequential number.

#### **Image/Component**

If you are displaying a list of images, enter the name of the Image file. The Image file must reside in the Images location as defined for the Data/Application Server in File Location in the Web Administrator.

If you are displaying Web components, this field will be entitled 'Component'. In this case, enter the name of the Web component to be displayed.

#### Description

This description is displayed when a mouse is positioned over the banner when it is viewed in the browser.

## URL

The URL of the supplier of the image. When you click on the image in the Banner Web component, the URL will be used to locate the Web site attached to the image. You will require the URL only if you are creating a list of Images.

## **Click Count**

LANSA for the Web provides you with a facility to keep count of the number of times the image is selected when the banner is displayed. This feature allows you to provide advertising space in your application.

Click-tracking only applies to Banner Web components that consist of images.

You can view the current click-tracking status of each individual item in a Banner Component by looking at its details in this Banner Detail dialog box. This number is automatically updated for each item each time the user clicks on a banner to visit that particular Web site / URL.

The click-tracking counter is held field W11CLK in LANSA table DC@W11.

↑ Banner Component (HTML mode)

# **File Component**

If you are adding or changing a File Web component, the Add/Change File Component dialog box, as appropriate, will be displayed. If you are creating a number of File Components, you can use the Duplicate feature.

Add File Component
Component: RMIT Description: File: Library:
Mode C Input C Output C Not Applicable
OK Cancel

### Description

Describes the component in the Web Component Registry.

## File and Library

The file definitions are specified in these entry fields. LANSA does not validate the existence of the file on the host. The file will not be created by LANSA. You must create the file.

The file is treated by LANSA as a data stream file. LANSA does not validate the content of the file. However, LANSA tags can be embedded in the file.

If you are connected to a host type Other then the Library entry field will be disabled and the File entry field will allow for a full path filename.

For the AS/400 files, if a library is not specified, the library list of the user profile at execution time will be used by LANSA to locate the file.

## Mode

You can choose to define the Web component as a mode dependent Web component by selecting the appropriate mode. For more details, refer to Web Components and Modes in the *Web Function Guide*.

10.6.1 Maintain Component.

## **Page Component**

If you have selected a Page as your Component Type, the Add or Change Page Component dialog box, as appropriate, is displayed. If you are creating a number of Page Components, you can use the Duplicate feature.

Add Page Component	
Component: WEBDOC Description:	
Page:	
Mode C Input C Output C Not Applicable	
OK Cancel	

### Description

Describes the component in the Web Component Registry.

## Page

Specify the name of the HTML/XML page. A Page Web component allows you to use an HTML/XML page as a Web component. The HTML/XML page must be created using this Web Function Editor. In other words, the HTML/XML page must be known to LANSA for the Web.

An example of a Page Web component is STDHEADER.

## Mode

You can choose to define the Web component as a mode dependent Web component by selecting the appropriate mode. For more details, refer to Web Components and Modes in the Web Function Guide.

10.6.1 Maintain Component

# Script Component (HTML mode)

If you are working with a Script Web component as your Component Type, the Add or Change Script Component dialog box, as appropriate, will be displayed. If you are creating a number of Script Components, you can use the Duplicate feature.

Add Script Component	
Component: WEBDOC Description:	
Script:	
Mode C Input C Output T Not Applicable	
OK Cancel	

## Description

Describes the component in the Web Component Registry.

## Script

Specify the name of the HTML page containing the script functions. Script Web components are treated by LANSA for the Web as HTML pages stored in the LANSA internal tables. This Script Web component page must be created using this Web Function Editor.

LANSA for the Web does not restrict you to a particular script language. In other words, you could be creating a JavaScript or VBScript Script Web Component.

LANSA for the Web does not validate the functions specified in the Script Web component. You must ensure that the script you have created is syntactically correct. You must verify your script before creating it as a Script Web component.

## Mode

You can choose to define the Web component as a mode dependent Web component by selecting the appropriate mode. For more details, refer to Web Components and Modes in the *Web Function Guide*.

10.6.1 Maintain Component.

## **Text Component**

If you are working with a Text Web component as your Component Type, the Add or Change Text Component dialog box, as appropriate, is displayed. If you are creating a number of Text Components, you can use the Duplicate feature.

Add Text Component
Component: WEBDOC Description: Text:
Mode C Input C Output OK Cancel

## Description

Describes the component in the Web Component Registry.

#### Text

Enter the text associated with the component. The length of the text is limited to 255 characters. The text may include HTML/XML tags. RDML tags as part of the text are not resolved.

#### Mode

You can choose to define the Web component as a mode dependent Web component by selecting the appropriate mode. For more details, refer to Web Components and Modes in the *Web Function Guide*.

10.6.1 Maintain Component.

# **Visual Component**

If you have selected a Visual Web component, the Add or Change Visual Component dialog box, as appropriate, is displayed. If you are creating a number of Visual Components, you can use the Duplicate feature.

Add Visual Co	mponent	
Component: Wi	EBDOC	
Description:		
Page:		
Mode		
Input		
OK	Cancel	1
	- 	

### Description

Describes the component in the Web Component Registry.

### Page

Specify the name of the HTML/XML page. Typically, the name of the page will be the same name as the Web component.

The named HTML/XML document must be created by the Web Function Editor.

#### Mode

This indicates the mode of operation for the Visual Web component. If the Visual Web component is to be used to replace an entry field, the mode should be set to Input. If you want the Visual Web component to replace an output field, select the Output option. For more details, refer to Web Components and Modes.

10.6.1 Maintain Component

# Web Link Component (HTML mode)

If you are working with a Web Link component as your Component Type, the Add/Change Web Link Component dialog box, as appropriate, is displayed. If you are creating a number of Web Link Components, you can use the Duplicate feature.

Add Web Link Component 🛛 🔀
Component: WEBDOC
Description:
Linked Process:
Linked Function:
Linked Description:
OnClick:
Type Button   Image
OK Cancel

#### Description

Describes the Web component in the Web Component Registry.

## **Linked Process and Linked Function**

The function to invoke when the link is selected is described by the Linked Process and Linked Function fields. These fields are not enabled if the OnClick check box is checked.

## **Linked Description**

This is the text displayed if the component is a button. If the component is an image, it is the text displayed when you position your mouse over the image/button when it is displayed in the browser.

## OnClick

If you check this box, you will be able to enter the commands to be executed in case of an OnClick event. You may have to edit DEFAULT\_SCRIPT or process name>\_SCRIPT to add the Java Script Function. If OnClick is

checked, the fields Linked Function and Linked Process will be disabled.

# Туре

The Type determines if the link is displayed as a button or an image. If it is to be displayed as an image, the image file name must also be supplied.

If an image is specified, the image file must be stored in the Images directory defined in the File Location page of the *Configure* ... options in the LANSA for the Web Administrator.

10.6.1 Maintain Component

# **10.6.2 Generate Component**

The Generate Component command allows you to either generate an input mode File Web component (File - HTML mode only) or an input mode Visual Web component (Visual).

LANSA for the Web supports the automatic creation of mode dependent Web components for:

- Drop downs
- List boxes
- Radio buttons
- Check boxes.
- Generate Visual Component
- Generate File Component (HTML mode)
- Considerations for using File Web Components
- 10.6 Components Menu

# **Generate Visual Component**

Visual Web components can be used to enhance the presentation of your functions.

When you select *Visual* ... from the Generate Component option, the Generate Visual Component dialog box is displayed.

Generate Visual Compone	nt 🛛 🔀
Component:	
Extension:	HTM 💌
Sec. Ext.:	
Visual Type:	Check box
Form Element Name Override:	
Repository Data	
File:	
Field for Value:	
Field for Description:	
🗖 Use Library List	
M Include blank entry	
Static Data	
Value:	
Description:	
D Coolipsion.	1
OK Cancel	

#### Component

Specify the name of the Web component.

#### Sec. Ext. (XML mode)

Secondary extension specifies the sub-extension (also called the XML Application) to be used to identify the Component. This value enables you to simplify the search for XML documents.

## Visual Type

Select the required Type from the drop down list.

## **Repository Data**

If you have selected a Visual Type of Drop Down, List Box or Radio Button,

you will need to provide Repository-based information. LANSA for the Web will build the visual Web component from the data contained in the file you specify.

# File

The name of the file. This file must be known to LANSA Repository on the host. The HTML/XML generated is based on the data in this file.

## **Field for Value**

The name of the field you want to use for the VALUE keyword in the resulting HTML/XML. This is the field containing the data that will be returned to your application at the Server.

## **Field for Description**

The description of the entry displayed to the user.

## **Use Library List**

Select this option if you want to use the user profile's library list to locate the source file instead of the library associated with the file in the LANSA Repository.

## Include blank entry

By default, a blank entry is generated which is used when evaluating the <RDML CHECKVALUE> tag. This sets the default field value to \*BLANK. If you do not want to generate a blank entry for the component, deselect this option.

# Static Data

If you are building a check box, you must enter the two values in this area.

# Value

The value for the checked state of the check box.

## Description

The description of the check box.

## Form Element Name Override

This is used for HTML only. If a name is provided it will be used as the name of the HTML form elements.

If not provided, the component name is used to name the form elements.

# OK

When you select the *OK* button, the Visual Web component will be generated

automatically and the Web Component Registry updated to reflect the Web component. The component will be created as an input Visual Web component.

If a Web component of the same name already exists, the new Visual Web component will replace the existing Web component. The HTML/XML associated with the existing Web component will be backed up before the new Web component is created.

**Note:** Make sure that you enter correct values for the fields in the Repository Data area, otherwise the component will not be generated.

10.6.2 Generate Component

Test Case:

## Test Plan Writer: Torkel CronholmDate Written:

## LANSA Version: 11.4Date Last Tested:

**Purpose:** Generate Visual Component from Web Function Editor dialog.

**Comments:** For all test cases it is assumed that the tester knows how to generate a visual component, and how to ensure that it is created. The test has to be performed on both a RDML and a RDMLX partition on both a Windows and an IBM i backend, ie, on four combinations (RDML Windows, RDML IBM i, RDMLX Windows, RDMLX IBM i). Some tests will not be applicable for RDML partitions, such as tests on LL2 files.

Use the option to provide a Form Element Name Override for some of the tests. Please indicate with an O) which test items this option was used on.

Use the option to provide a Use Library List for some of the tests. Please indicate with an L) which test items this option was used on.

Use the option to provide a Include blank entry for some of the tests. Please indicate with a B) which test items this option was used on.

Some of the tests should generate a web component where there already exists a web component. Please indicate with an R) which test items this test was done for.

For this test, suggested naming convention for components is iiiEVXY#LD,

where

Iii stands for the initials of the tester

E stands for Editor (the components can be generated by a BIF as well as from command line)

V is for a Visual component

X is the type of component. Use C for Check box, D for Drop down, L for ListBox and R for Radio button.

Y is for form element name override. Use Y for Yes and N for No.

*#* is denoting LL1 or LL2 file where applicable. Use 1 for LL1, 2 for LL2 and 0 if not applicable).

L is for data type for Value when creating a Drop down, ListBox or Radio button. Use A for Alpha, P for Packed, S for Signed, C for Char, D for Date, T for Time, Z for DateTime, I for Integer, F for Float, S for String and omit if not applicable.

D is for data type for Description when creating a Drop down, ListBox or Radio button. Use A for Alpha, P for Packed, S for Signed, C for Char, D for Date, T for Time, Z for DateTime, I for Integer, F for Float, S for String and omit if not applicable.

Example: A tester named Douglas Noel Adams creating a Visual component using a Drop down with element override for an LL2 file (using Alpha for Value and Signed for Description) should name the component: DNAEVDY2AS. Always name the form element name override in a consistent manner, ie, add \_O to the name to create the override. Form Element Name Override would be DNAEVDY2AS\_O for this example.

For Check Box, use any test for Value and Description fields.

In test items 1 - 27 the tester will attempt create a visual component according to above.

The expected result in test items 1 - 27 is that the component is created as per the

test.

Tes	ter:	_Date Commenced:	Date
Co	mpleted:		
	Test	Expected Results	Pass/ Fail
1	Checkbox:		
2	Drop down, ListBox and Radi using LL2 file	o button	
	Value: Alpha, Description: Al	pha	
3	Drop down, ListBox and Radi using LL2 file	o button	
	Value: Packed, Description: A	lpha	
4	Drop down, ListBox and Radi using LL2 file	o button	
	Value: Signed, Description: A	lpha	
5	Drop down, ListBox and Radi using LL2 file	o button	
	Value: Char, Description: Alpl	าล	
6	Drop down, ListBox and Radi using LL2 file	o button	

Value: Date, Description: Alpha

7 Drop down, ListBox and Radio button using LL2 file

Value: Time, Description: Alpha

8 Drop down, ListBox and Radio button using LL2 file

Value: DateTime, Description: Alpha

9 Drop down, ListBox and Radio button using LL2 file

Value: Integer, Description: Alpha

10 Drop down, ListBox and Radio button using LL2 file

Value: Float, Description: Alpha

11 Drop down, ListBox and Radio button using LL2 file

Value: String, Description: Alpha

12 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Alpha

13 Drop down, ListBox and Radio button

using LL2 file

Value: Alpha, Description: Packed

14 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Signed

15 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Char

16 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Date

17 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Time

18 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: DateTime

19 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Integer

20 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Float

21 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: String

22 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Alpha

23 Drop down, ListBox and Radio button using LL1 file

Value: Packed, Description: Alpha

24 Drop down, ListBox and Radio button using LL1 file

Value: Signed, Description: Alpha

25 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Alpha

26 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Packed	
27 Drop down, ListBox and Radio button using LL1 file	
Value: Alpha, Description: Signed	
28 Drop down, ListBox and Radio button using incorrect LL1 file	Error message in Web Function Editor
29 Drop down, ListBox and Radio button using incorrect LL2 file	Error message in Web Function Editor
30 Drop down, ListBox and Radio button using correct LL1 file, but incorrect fieldnames	No message in Web Function Editor, but the components will not be created
31 Drop down, ListBox and Radio button using correct LL2 file, but incorrect fieldnames	No message in Web Function Editor, but the components will not be created
32 Leave some of the required text fields empty when generating a visual component	Error message in Web Function Editor
33 Check the tab order in the dialog (Either when in "Check Box" mode or in "Drop down, ListBox and Radio" mode)	Tab order is from top to bottom, left to right

# Generate File Component (HTML mode)

When you select *File*... from the Generate Component option, the Generate File Component dialog box is displayed. Refer to Considerations for using File Web Components.

Generate File Component
Component:
Extension:
Sec. Ext.:
Visual Type: Check box
Repository Data
Field for Value:
Field for Description:
☐ Use Library List ☑ Include blank entry
Static Data Value:
Description:
File (out file):
Use library in component definition
OK Cancel

Most of the fields in this dialog box are identical to the Generate Visual Component dialog box.

#### Component

Specify the name of the Web component.

#### Sec. Ext. (XML mode)

Secondary extension specifies the sub-extension (also called the XML Application) to be used to identify the Component. This value enables you to simplify the search for XML documents.

#### **Visual Type**

Select the required Type from the drop down list.

## **Repository Data**

If you have selected a Visual Type of Drop Down, List Box or Radio Button, you will need to provide Repository-based information. LANSA for the Web will build the visual Web component from the data contained in the file you specify.

## File

The name of the file. This file must be known to LANSA Repository on the host. The HTML/XML generated is based on the data in this file.

## **Field for Value**

The name of the field you want to use for the VALUE keyword in the resulting HTML/XML. This is the field containing the data that will be returned to your application at the Server.

#### **Field for Description**

The description of the entry displayed to the user.

#### **Use Library List**

Select this option if you want to use the user profile's library list to locate the source file instead of the library associated with the file in the LANSA Repository.

## Include blank entry

By default, a blank entry is generated which is used when evaluating the <RDML CHECKVALUE> tag. This sets the default field value to \*BLANK. If you do not want to generate a blank entry for the component, deselect this option.

## **Static Data**

If you are building a check box, you must complete these two entries:

## Value

The value for the checked state of the check box.

#### Description

The description of the check box.

## File (out file)

Enter the target file specification. This file is created as a source physical file and holds the generated HTML for the Web component.

#### Use library in component definition

Select this option if you want to use the user profile's library list to locate the source file instead of the library associated with the file in the LANSA Repository. This option is disabled if you are connected to a host type Other. If this option is selected the library and file combination is used each time the component is required in your application.

# OK

When you select the *OK* button, the File component will be generated automatically and the Web Component Registry will be updated to reflect the new/modified component. The component will be created as an input File component.

**Note:** Make sure that you enter correct values for the fields in the Repository Data area, otherwise the component will not be generated.

If a Web component of the same name already exists, the new Visual Web component will replace the existing Web component. The HTML/XML associated with the existing Web component will be backed up before the new Web component is created.

10.6.2 Generate Component

# **Considerations for using File Web Components**

LANSA will automatically generate an empty AS/400 source physical file for your AS/400 file Web components.

Using the Web Function Editor, select the Components menu and choose the Generate Component - File.

You will need to enter the target AS/400 file specification (library name/file name) in the AS/400 file (out file) field. This file will be created as a source physical file and will be used to hold the "data", i.e. the HTML for the Web component.

If the Use library in component definition option is selected, each time the component is required in your application, the library and file combination is used. Do not select the option if you want to use the user profile used by LANSA for the Web to locate the file when you run your application. In this case, the library specification of the file is not stored in the LANSA for the Web component registry. This option is very useful if your application requires you to display different data for the same field, depending on the user profile used to run the application. You can use a different AS/400 file Web component for each user. In this case, a different AS/400 file would be created, each containing the HTML to display the data, for each user profile. These AS/400 files would be installed in the appropriate AS/400 library. The library list attached to the user profile would then be used to locate the correct file.

If the Use library in component definition option is not selected, the library for this component will not be stored as part of the file definition in the Web component registry. When the application is executed, LANSA for the Web will still use the Web component, but, it will use the library list attached to the user profile in order to locate the AS/400 physical file (since no library is attached to the file definition in the Web component registry).

10.6.2 Generate Component

Test Case:

## Test Plan Writer: Torkel CronholmDate Written:

## LANSA Version: 11.4Date Last Tested:

Purpose: Generate File Component from Web Function Editor dialog.

**Comments:** For all test cases it is assumed that the tester knows how to generate a file component, and how to ensure that it is created. The test has to be performed on both a RDML and a RDMLX partition on both a Windows and an IBM i backend, ie, on four combinations (RDML Windows, RDML IBM i, RDMLX Windows, RDMLX IBM i). Some tests will not be applicable for RDML partitions, such as tests on LL2 files.

Use the option to provide a Use Library List for some of the tests. Please indicate with an L) which test items this option was used on.

Use the option to provide a blank entry for some of the tests. Please indicate with a B) which test items this option was used on.

Some of the tests should generate a web component where there already exists a web component. Please indicate with an R) which test items this test was done for.

On IBM i ONLY, some of the tests should generate a web component where Use library in component definition is selected. Please indicate with a U) which test items this test was done for.

1) if you specify the name of the output file, eg, FILE123, it will be created in the current library

2) if you specify the name of the output file, eg, P01DTALIB\FILE123, it will be created in P01DTALIB. If you ticked the check box the component definition will also include the library so the web runtime knows where to get the file

3) if you specify the name of the output file, eg, P01DTALIB\FILE123, it will be created in P01DTALIB. If you DID NOT tick the check box the component definition will NOT include the library so the web runtime will use the library list to get the file

For this test, suggested naming convention for components is iiiEFX#LD, where

iii stands for the initials of the tester

E stands for Editor (the components can be generated by a BIF as well as from command line)

F is for a Visual component

Test

X is the type of component. Use C for Check box, D for Drop down, L for ListBox and R for Radio button.

*#* is denoting LL1 or LL2 file where applicable. Use 1 for LL1, 2 for LL2 and 0 if not applicable).

L is for data type for Value when creating a Drop down, ListBox or Radio button. Use A for Alpha, P for Packed, S for Signed, C for Char, D for Date, T for Time, Z for DateTime, I for Integer, F for Float, S for String and omit if not applicable.

D is for data type for Description when creating a Drop down, ListBox or Radio button. Use A for Alpha, P for Packed, S for Signed, C for Char, D for Date, T for Time, Z for DateTime, I for Integer, F for Float, S for String and omit if not applicable.

Example: A tester named Douglas Noel Adams creating a File component using a Drop down with element override for an LL2 file (using Alpha for Value and Signed for Description) should name the component: DNAEFD2AS.

For Check Box, use any test for Value and Description fields.

Ensure that the nominated file is physically created in the File System for both Windows and IBM i. Also ensure that the nominated file overwrites the file if it already exists. An easy way to do the overwrite test is to generate a different type of component and check its contents afterwards.

In test items 1 - 27 the tester will attempt create a file component according to above.

The expected result in test items 1 - 27 is that the component is created as per the test.

Tester:	_Date Commenced: _	Date
Completed:		

Expected Results Pass/ Fail

- 1 Checkbox:
- 2 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Alpha

3 Drop down, ListBox and Radio button using LL2 file

Value: Packed, Description: Alpha

4 Drop down, ListBox and Radio button using LL2 file

Value: Signed, Description: Alpha

5 Drop down, ListBox and Radio button using LL2 file

Value: Char, Description: Alpha

6 Drop down, ListBox and Radio button using LL2 file

Value: Date, Description: Alpha

7 Drop down, ListBox and Radio button using LL2 file

Value: Time, Description: Alpha

8 Drop down, ListBox and Radio button using LL2 file

Value: DateTime, Description: Alpha

9 Drop down, ListBox and Radio button using LL2 file

Value: Integer, Description: Alpha

10 Drop down, ListBox and Radio button using LL2 file

Value: Float, Description: Alpha

11 Drop down, ListBox and Radio button using LL2 file

Value: String, Description: Alpha

12 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Alpha

13 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Packed

14 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Signed

15 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Char

16 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Date

17 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Time

18 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: DateTime

19 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Integer

20 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: Float

21 Drop down, ListBox and Radio button using LL2 file

Value: Alpha, Description: String

22 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Alpha

23 Drop down, ListBox and Radio button using LL1 file

Value: Packed, Description: Alpha

24 Drop down, ListBox and Radio button using LL1 file

Value: Signed, Description: Alpha

25 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Alpha

26 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Packed

27 Drop down, ListBox and Radio button using LL1 file

Value: Alpha, Description: Signed

28 Drop down, ListBox and Radio button using incorrect LL1 file	Error message in Web Function Editor
29 Drop down, ListBox and Radio button using incorrect LL2 file	Error message in Web Function Editor
30 Drop down, ListBox and Radio button using correct LL1 file, but incorrect fieldnames	No message in Web Function Editor, but the components will not be created
31 Drop down, ListBox and Radio button using correct LL2 file, but incorrect fieldnames	No message in Web Function Editor, but the components will not be created
32 Leave some of the required text fields empty when generating a visual component	Error message in Web Function Editor
33 Check the tab order in the dialog (Either when in "Check Box" mode or in "Drop down, ListBox and Radio" mode)	Tab order is from top to bottom, left to right

# 10.6.3 Graphic Variables

For more details about graphic variables, refer to Graphic Variables in the *Web Function Guide*.

The Graphic Variables command allows you to Add, Change or Delete graphic variables.

raphic Variables			
Name	Туре	Value	^
🐎 *LW3BL_WIZ_LIST2	Text	*NOIB	
👧 *LW3BL_WIZCSS_01	Text	*NOBI	
👧 *LW3BL_WIZCSS_02	Text	*NOBI	
👧 *LW3BL_WIZCSS_03	Text	*NOBI	
👧 *LW3BL_WIZMENU	Text	*NOBI	
👧 *LW3BL_WIZQST	Text	*NOBCI	
👧 *LW3BLACB_WIZ_LIST2	Color	DDDDDD	
👧 *LW3BLACB_WIZCSS_01	Color	DDDDDD	~
	<u> </u>	000000	
Close Add	Char		elete

Select a Graphic Variable in the list and press the *Delete* button to delete a Graphic Variable. You will be asked to confirm your deletion.

Select a Graphic Variable in the list and press the *Change*... button to open the Change Graphic Variable dialog box.

Press the *Add*... button to Add Graphic Variable, or use the Duplicate feature.

# Duplicate

If you select a graphic variable and click the right mouse button, a floating menu is displayed. On this menu are menu items: Change, Delete, Duplicate. If you select Duplicate, the selected graphic variable will be used as a template for a new component and its details will be displayed in the Add Graphic Variable dialog box.

With this feature you can easily create multiple graphic variables without having to re-enter the values.

# Task Tracking

If you are using LANSA's Task Tracking, then you will be asked for the Task Id when you add, change, or delete a Graphic Variable.

# 10.6 Components Menu

# Add Graphic Variable

For more details about graphic variables, refer to Graphic Variables.

Add Grap	hic Variable 🛛 🔀
Name:	
Туре:	Color
Value:	
ОК	Cancel

#### Name

Enter the name of this Graphic Variable. Note that the name must start with an asterisk (\*). If you have used the Duplicate feature, the name displayed will be the name of the graphic variable you selected as a template for this new graphic variable. In this case, you must change this name.

If the Graphic Variable displayed is not the one on which you wanted to base the new graphic variable, press Cancel to return to the Graphic Variables list and select the correct one.

#### Туре

Select the Type from the drop down menu. Valid types include Color, File or Text. For more details, refer to Types of Graphic Variables.

#### Value

Enter the value that you require for the Type selected. A maximum of 255 characters can be entered

If you are specifying an image file name, be sure that the specified image is stored in the image location as defined using the LANSA for the Web Administrator.

If you are specifying a color, you can use the name (such as limegreen) or the hexadecimal value (such as #32CD32).

If you are specifying text, you can use HTML/XML tags in the text. Do not use RDML tags in the text.

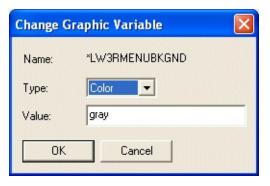
## OK

Press the *OK* button when done.

10.6.3 Graphic Variables

# **Change Graphic Variable**

For details refer to Graphic Variables.



#### Name

You cannot change the name. If the Graphic Variable displayed is not the one you want to change, press *Cancel* to return to the Graphic Variables list and select the correct one.

#### Туре

Select the Type from the drop down menu. Valid types include Color, File or Text. For more details, refer to Types of Graphic Variables.

#### Value

Enter the value that you require for the Type selected. A maximum of 255 characters can be entered

If you are specifying an image file name, be sure that the specified image is stored in the image location as defined using the LANSA for the Web Administrator.

If you are specifying a color, you can use the name (such as limegreen) or the hexadecimal value (such as #32CD32).

If you are specifying text, you can use HTML/XML tags in the text. Do not use RDML tags in the text.

## OK

Press the *OK* button when done.

10.6.3 Graphic Variables

# **10.7 Options Menu**

The Options Menu contains the following options:

- **10.7.1 Configure** to set the options that control your use of the Web Function Editor.
- 10.7.2 Font a standard windows Font dialog box.
- 10.7.3 Invoke Third Party Editor use this command to start the third party editor, if you using one.

# 10.7.1 Configure

The Configure command allows you to set the options you require when you are using the Web Function Editor. These are:

Third Party Editor

View

Miscellaneous

The pages you use to specify your preferences will vary slightly depending on whether you are working in HTML or XML mode. If you have been accustomed to working with HTML pages and now you want to modify XML documents, you should review your configuration options to see check if they are still suitable while working in XML mode.

10.7 Options Menu

# **Third Party Editor**

This page allows you to specify the HTML or XML Editor of your choice. You do not have to use a third party editor.

Configuration Options	×
Third Party <u>E</u> ditor │ <u>V</u> iew │ Miscellaneous │	
Editor: notepad.exe Browse	
Start Editor every time Automatically save changes after returning from Editor	
OK Cancel Help	

#### Editor

The name (including the path) of the editor you want to use. Press the Browse button to locate the Editor on your PC.

You must not edit the LANSA generated HTML/XML using a graphical editor even though the editing tool you use may allow you to. This is because the generated HTML/XML contains LANSA Web components (some of them are also HTML/XML pages). In addition, there are LANSA tags embedded. For example, you cannot edit the <RDML MERGE="&BUTTONS"> tag graphically since this is an instruction to LANSA when the Web function is executed.

## Start Editor every time

Select this option if you want to start the editor (that you have specified here) every time you use the Open command to retrieve an HTML/XML page from the LANSA Repository.

If this option is not selected, you can start your chosen editor by selecting Invoke Editor from the Options menu.

## Automatically save changes after returning from Editor

When you have finished editing the HTML/XML with your chosen editor, the

LANSA Web Function Editor will check if any changes have been made to the HTML/XML. If there have been changes and if you have selected this option, the HTML/XML is saved into the LANSA Repository.

If this option is not selected, you will have to save the changes manually by selecting the Save command in the File menu.

10.7.1 Configure

# View

This page allows you to choose the features to be used when you compare text.

Configuration Options	×
Third Party Editor View Miscellaneous	
Compare Background Colors	
Inserted Text Kp>	
Deleted Text <b>xpxx/px</b> 👻	
, <b></b>	
Edit and Compare Windows	
Synchronized Scrolling	
Horizontal Split	
Vertical Split	
OK Cancel Help	

## **Compare Background Colors**

Define the colors to be used for the Inserted Text and Deleted Text.

# **Edit and Compare Windows**

Select the Synchronize Scrolling option to synchronize the scrolling between the two windows when comparing two HTML/XML pages.

The comparison windows can be split Horizontally or Vertically.

# 10.7.1 Configure

# Miscellaneous

This page allows you to control the Most Recently Used list in the File menu.

Configuration Options	×	
Third Party Editor View Miscellaneous		
Most Recently Used Files Options Items in MRU list: 4 (Range 0-10) Clear MRU List		
Max. Undo Actions: 50 (Range 1-32000)		
OK Cancel Help		

## Items in MRU list

Enter the number of items, in the range 0 to 10, that you would like retained for listing in the File menu.

## **Clear MRU List**

Press this button to clear the current Most Recently Used list.

## Max. Undo Actions

Specify the number of actions you wish to retain so that you can undo them if necessary. To undo actions, select Undo from the Edit menu.

Note that the higher the number you enter here, the more system resources that will be used.

## Enable archive functionality when saving

This option will only be enabled if the Administrator options have been set: Enable automatic backup of generated HTML and XML and

the Save previous copies is greater than zero.

If the box is checked, you will be asked if you wish to archive the previous version of the file whenever a document is saved. If you answer Yes when prompted, LANSA will save the original version of the document as Version 1.

For more details, refer to Versioning of Pages. ↑ 10.7.1 Configure

# 10.7.2 Font

The Font command allows you to change the font settings in the Editor. It does not change the font settings of the HTML/XML lines in the HTML/XML page. If you need to change the font setting of a particular HTML/XML line, use the <FONT> tag.

10.7 Options Menu

# **10.7.3 Invoke Third Party Editor**

This command invokes the HTML/XML editor you have chosen to edit your HTML/XML pages. This command is disabled if you have not chosen an editor.

10.7 Options Menu

# 10.8 Tools Menu

The Tools Menu contains the following options:

- 10.8.1 Keywords
  - Add Keyword
  - Maintain Keyword

# 10.8.1 Keywords

Keywords allow you to specify the links to other functions for WEBEVENT functions. The links are identified by keywords, which are the values assigned to the user defined keys (i.e. enter, continue, and so on) in your LANSA functions.

The Keyword submenu from the Tools menu allows you to choose either Add or Maintain. You can create a new keyword from either command, however, the Add option is quicker if you simply want to create a new keyword - it saves you waiting for the list of existing keywords to be displayed.

You cannot change a keyword. First you must delete the link, then add a new keyword.

Alternatively, you can use Web link components to define the links instead of using this command.

Add Keyword

Maintain Keyword

10.8 Tools Menu

# Add Keyword

If you are creating a new keyword, select *Add* from the Keyword sub menu. The Add Keyword dialog box is displayed.

This dialog box allows you to define the links to other LANSA functions for your user defined keys.

Add Keyword		<
Process: Function: Keyword:	Language: Description: ENG  Linked Process: Linked Function:	
	DK Cancel	

#### **Process, Function and Keyword**

Specify the LANSA process, function and keyword respectively.

The keyword is case sensitive. If you use 'SUBMIT' in uppercase in your RDML USER\_KEY statement, then you must use 'SUBMIT' and not 'submit' when defining the keywords.

#### Language

Select the language for the button's description (this option is only enabled if you are connected to a multilingual partition).

#### Description

Specify the description for the button on the browser window.

#### Linked Process and Linked Function

Specify the linked function and Function.

10.8.1 Keywords

#### **Maintain Keyword**

To change some keyword details, or to delete a keyword, you need to select it from a list of Processes. To limit the list you need to select from, when you select the Maintain command, a Process dialog box is displayed for you to enter a full or partial Process name.



#### Process

To limit the number of LANSA processes listed, enter a partial string sequence, for example, "DEJ" to display all LANSA processes starting with "DEJ".

To display all the keywords in your system, leave this field blank.

When you select the *OK* button, the Maintain Keywords dialog box is displayed listing all the processes matching the criteria that you have requested.

Process	Function	Keyword	Description	Linked Process	Linked F
WIZARD2 WIZARD2 WIZARD2 WIZARD2	WIZE04 WIZL02 WIZL02 WIZM04	Wizsav Wizfin Wiznxt Wizsav	Apply Finish Next Apply	WIZARD2 WIZARD2 WIZARD2 WIZARD2	WIZE04 WIZL02 WIZL02 WIZM04
<					

The list shows you the processes and functions to which the keywords belong. The Keyword parameter is the value used to define the USER\_KEYS parameter in the DISPLAY or REQUEST command in your RDML function.

To change a keyword, select a Process in the list and press the *Change*... button. The Change Keyword dialog box is displayed.

To delete a keyword, select a Process in the list and press the *Delete* button. You will be asked to confirm your deletion.

To add a new keyword, press the Add button. The Add Keyword dialog box is displayed. Alternatively, to add a new keyword, you can select New from the Keyword command.

↑ 10.8.1 Keywords

# **Change Keyword**

If you are changing a keyword, select a process in the list in the Maintain Keywords dialog box and press the *Change* button. The Change Keyword dialog box is displayed.

This dialog box allows you to define the links to other LANSA functions for your user defined keys.

Change Keyword		X
Process: WIZARD2 Function: WIZM04 Keyword: WIZSAV	Language: Description: ENG Apply Linked Process: WIZARD2 Linked Function: WIZM04	
ОК	Cancel	

#### **Process, Function and Keyword**

These are as specified for the Process you have chosen.

The keyword is case sensitive. If you use "SUBMIT" in uppercase in your RDML USER\_KEY statement, then you must use "SUBMIT" and not "submit" when defining the keywords.

#### Language

The language for the button's description (this option is only enabled if you are connected to a multilingual partition).

#### Description

The description for the button displayed in the browser window.

#### Linked Process and Linked Function

The linked function and Function.

Press *OK* to accept your changes or *Cancel* to dismiss this dialog box.

↑ Maintain Keyword

Test Case:

#### Test Plan Writer: Torkel CronholmDate Written:

#### LANSA Version: 11.4Date Last Tested:

**Purpose:** Test that the help file is loaded correctly. The help file is loaded with the same language as the program is running in. The name of the help file is LWEdt<LANG>.chm where <LANG> is either Eng, Fra of Jpn. Note that at the time of writing, no LWEdtFra.chm or LWEdtJpn.chm file exists. Test for those has to be simulated by using copies of other files. The help file(s) are located in the WebUtilities folder of the configuration.

You can choose which language to run the program in by editing the registry setting responsible for the language, ie, HKEY\_LOCAL\_MACHINE\SOFTWARE\LANSA\<encoded name of path>\LWEBGEN. Item Language is either ENG, FRA or JPN

The French language DLL is named lwebefra.dll and is located in the WebUtilities folder of the configuration.

The Japanese language DLL is named lwebejpn.dll and is located in the WebUtilities folder of the configuration.

#### **Comments:**

Tester:	Date Commenced:	Date
Completed:		

	Test	Expected Results	Pass/ Fail
1	Ensure you have a copy of the Eng help file in the same folder as the program.	File present there after 11.4 install.	
2	Start the program in English. Use menu to go to Help - Contents	The Eng help file is loaded	

3	Start the program in French. Ensure there is no LWEdtFra.chm file present. Use menu to go to Help - Contents	The Eng help file is loaded
4	Start the program in French. Ensure there is no LWEdtJpn.chm file present. Use menu to go to Help - Contents	The Eng help file is loaded
5	To test language version of the help file, you can copy CHM files from elsewhere and name them LWEdtFra.chm resp. LWEdtJpn.chm	N/A
6	Start the program in English. Use menu to go to Help - Contents	The Eng help file is loaded
7	Start the program in French. Ensure there is a LWEdtFra.chm file present. Use menu to go to Help - Contents	The "Fra help file" is loaded
8	Start the program in Japanese. Ensure there is a LWEdtJpn.chm file present. Use menu to go to Help - Contents	The "Jpn help file" is loaded
9	To test that the language DLL and the help file go hand in hand, we need to simulate that the chosen language DLL is missing.	N/A
10	Rename the FRA language DLL.	N/A
11	Start the program in French. Ensure there is no LWEdtFra.chm file present. Use menu to	Program will start in English and

	go to Help - Contents	Eng help file is loaded
12	Start the program in French. Ensure there is a LWEdtFra.chm file present. Use menu to go to Help - Contents	Program will start in English and Eng help file is loaded
13	Rename the FRA language DLL back to what is what before step 10	N/A
14	Rename the JPN language DLL.	N/A
15	Start the program in Japanese. Ensure there is no LWEdtJpn.chm file present. Use menu to go to Help - Contents	Program will start in English and Eng help file is loaded
16	Start the program in Japanese. Ensure there is a LWEdtJpn.chm file present. Use menu to go to Help - Contents	Program will start in English and Eng help file is loaded
17	Rename the JPN language DLL back to what is what before step 14	N/A
18		

#### Test Case:

#### **Test Plan Writer: Torkel CronholmDate Written:**

#### LANSA Version: 11.4Date Last Tested:

**Purpose:** Test that the About Box has the correct information

#### **Comments:**

 Tester:
 \_\_\_\_\_Date Commenced:
 \_\_\_\_\_Date

 Completed:
 \_\_\_\_\_\_

	Test	Expected Results	Pass/ Fail
1	Start the program. Cancel from the connect screen. Do Help About.	"LANSA Editor Mode Version 11.4.0 (11404) © 2007 LANSA"	
2	Exit the program	Program exits	
3	Start the program and connect in HTML mode.	"LANSA Editor HTML Mode Version 11.4.0 (11404) © 2007 LANSA"	

4	Exit the program	Program exits
5	Start the program and connect in XML mode.	"LANSA Editor XML Mode Version 11.4.0 (11404) © 2007 LANSA"
6	Exit the program	Program exits
7	Start the program and connect in BASIC mode.	"LANSA Editor BASIC Mode Version 11.4.0 (11404) © 2007 LANSA"
8	Exit the program	Program exits
9		

# 12. Web Development Tips & Techniques

Following are some general Web development tips and techniques for building your Web Function Applications:

- 12.1 HTML as Field Contents
- 12.2 Automatic Data Apportionment
- 12.3 Considerations for Browse Lists
- 12.4 Message Presentation Layout
- 12.5 Using <RDML MERGE="&END">
- 12.6 Cascading Style Sheets
- 12.7 Retrieve Additional Information from Browse List
- 12.8 Handle the ENTER key in Browsers
- 12.9 Embed a Calendar Control
- 12.10 Modify LANSA for the Web Messages
- 12.11 Set the Initial Focus in an HTML Page
- 12.12 Tailoring the DEFAULT\_SCRIPT
- 12.13 Generate Static Page Output to the IFS (Integrated File System)
- 12.14 Integrate LANSA Applications with Static HTML Pages
- 12.15 CheckBox Visual Web Components
- 12.16 Extend LANSA Drop Downs

#### 12.17 Modifying charset for non-English Systems

It is strongly recommended that you review the Web and Internet examples in the LANSA SET (Samples Examples Templates) materials. These materials are available on the LANSA Encyclopedia CD-ROM and the www.lansa.com Web site. You might find the following types of samples and examples:

- Select Multiple Entries in List
- Checkbox Initial State
- Data Apportionment
- Buttons Linking to Same Function
- Determining Selected Row Number
- Expandable Menus
- Select a Date from a Calendar

- Select a Start and End Date from a Calendar
- Select a Date in Browse list
- Input/Output Fields in Browse list Column
- Input/Output Fields in Browse list Row
- Downloading a File from IFS
- Uploading a File to IFS
- Everything about Browse lists
- Handling Enter Key Problems

## **12.1 HTML as Field Contents**

A technique that can be used to enhance your application is to embed the HTML as the contents of one or more fields in your application. The HTML will be accepted as browser instructions.

For example, if you wanted to embed hyperlinks into your application, the HTML for the hyperlink could be the value of a particular field. The field could be a working field in your RDML function. The value of the field can be assigned programmatically.

For example, the DEPTAB file contains a Department Code and Department Description. If you were to use a Department Description as follows:

```
<center>Adminstration Department</center>
```

then the HTML tags would be processed by the browser and only the data would be displayed as follows:

Administration Department

The Department Description would appear centered in the display.

## **12.2 Automatic Data Apportionment**

LANSA for the Web supports the automatic apportionment of data. This feature is particularly useful for fields in your HTML form where the data can exceed the 256-character limitation. For example, if you have a field in your HTML form that accepts user feedback or comments, you would require this field to accept more than 256 characters. However, LANSA fields are limited to 256 characters.

LANSA for the Web allows you to use a single field in your HTML form, but yet if the data exceeds the field's length, the data will not be lost.

The technique used by LANSA for the Web is to use sibling fields, following the naming convention:

<field name><sequence number>

The name of the sibling fields consists of two parts, the field name and a sequence number. The length of the sibling field name must be 9 characters, with the sequence number part padded with leading zeros.

For example, if your field name is COMMENT. The sibling fields would be COMMENT01, COMMENT02, up to COMMENT99. However, if the field name were COMMT, the sibling fields would be COMMT0001, COMMT0002, up to COMMT9999.

In other words, if the data length you intend to support is large, then the field name should be short, to allow for more sibling fields.

If the sibling fields are used in your application, LANSA for the Web will apportion the data and store each part of the divided data in the sibling fields. If the number of fields is too small to hold all of the data, the rest of the data will be lost.

As an example, if you wanted to create a component for the COMMENT field that allows the user to type in as much comments/feedback as he wishes, you can create a textarea component. This component can be created as an input mode Visual Web component.

```
<RDML CHECKVALUE="YES">
<textarea name="COMMENT" rows="6" cols="75">
<RDML MERGE="COMMENT">
<RDML MERGE="COMMENT01">
<RDML MERGE="COMMENT02">
<RDML MERGE="COMMENT03">
```

```
<RDML MERGE="COMMENT04">
<RDML MERGE="COMMENT05">
</textarea>
</RDML>
```

You will notice that the field is only identified by a single name, COMMENT. However, the data is populated using 6 fields, COMMENT, COMMENT01, COMMENT02, COMMENT03, COMMENT04 and COMMENT05, using the <RDML MERGE> LANSA tag.

When the form is submitted to the Web server, LANSA for the Web will handle the apportionment of the data automatically. It determines if the length of the returned data is greater than 256 characters. If the length exceeds 256 characters, it will apportion the data into portions, with each portion split at 256 characters. The first 256 characters is stored in the COMMENT field. The second portion is stored in COMMENT01, the third in COMMENT02, and so on.

If the data length exceeds 1536 characters (256x6), then any data greater than 1536 characters will be lost, unless you have defined other sibling fields in your application, i.e. COMMENT06, etc.

Refer to 12.2.1 Specifying Apportionment Position.

## **12.2.1 Specifying Apportionment Position**

By default, LANSA for the Web uses 256 as the apportionment length. This is useful if your fields are defined to be 256 characters. This default value is not useful if your field length is less than 256 characters.

For such fields, LANSA for the Web allows you to specify the apportionment position. If a apportionment position is specified, LANSA for the Web will automatically apportion the data using the length specified.

To specify a apportionment position, the name attached to the field is modified, using the following naming convention:

<field name>-Lnnn

where <field name> is the name of the field, padding with trailing blanks to 10 characters. This means that the eleventh (11<sup>th</sup>) position of the field name must be the '-' character. The next character must be 'L', denoting apportionment length.

nnn is the desired apportionment length.

Taking the example above, if the field lengths of COMMENT, COMMENT01, etc. was 75 characters, then the HTML line defining the text area would be modified to be:

<textarea name="COMMENT -L075" rows="6" cols="75">If an apportionment length is specified as part of the name of the text area, this apportionment length will be used by LANSA for the Web to automatically apportion the data.

If you are using large text areas in WEBEVENT functions, you may need to enable Extended Exchange.

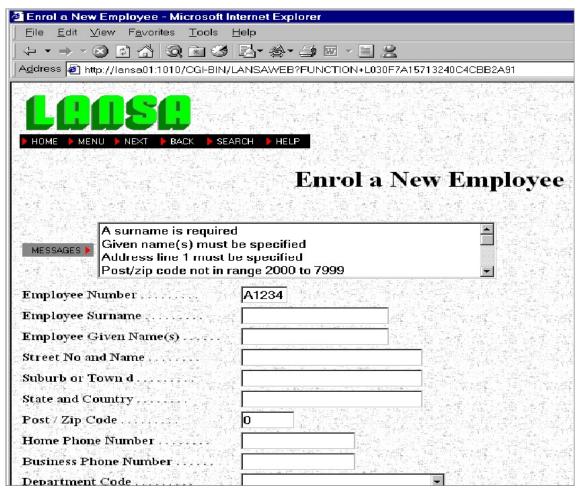
#### **12.3 Considerations for Browse Lists**

When working with a browse list for a LANSA Web function application, you must adapt your thinking to the browser paradigm. Consider some of the following points.

- What works in the 5250 environment may not work on the Web.
- Extending the browse list makes it longer and longer in LANSA for the Web.
- The concept of "page at a time" in the 5250 environment is not the same for the browser.
- You should consider building browse lists to contain just ONE page and work with the contents of that single page as it is displayed to the user.
- Consider using "position to" logic so that only the relevant records are retrieved and displayed. Using "position to" logic is a much better solution than having the user either scroll through huge lists or page forward through screen after screen.
- Also, WEBEVENT functions do not support the use of a list with \*SELECT field.

#### **12.4 Message Presentation Layout**

LANSA for the Web allows you to customize the presentation of LANSA messages in your application. By default, the LANSA messages are presented in a list box style.



The Message Presentation layout standard page is only supported at the process level. It is not supported at the function level.

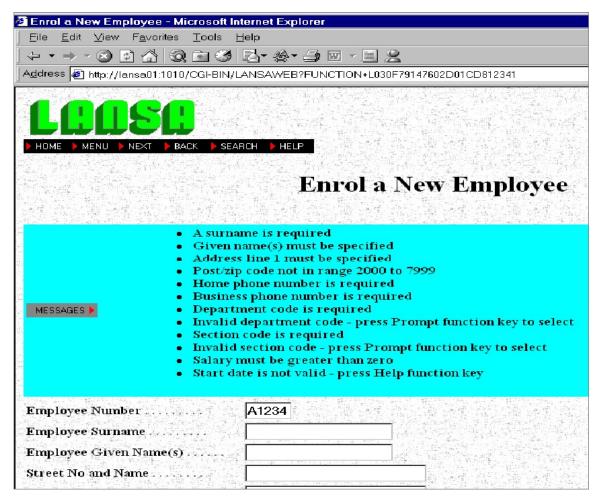
You can override the format of the message presentation by using the standard page feature. This is achieved by creating a standard page named as DEFAULT\_MSGPRES. In this standard page, you can define your own message presentation layout. If this page is defined, LANSA for the Web will use layout defined in this page to present the LANSA messages instead of the default format (using a list box).

The DEFAULT\_MSGPRES standard page is not shipped with the product. This standard page is a facility that allows you to override the default message

presentation format.

An example of a DEFAULT\_MSGPRES is shown below. This replaces the list box format with a list of messages:

```
<img src="
<RDML MERGE="*LW3IMGMESSAGES">" alt="Messages" border="0" />
<b><RDML MESSAGES></b>
```



LANSA Web functions use a LANSA tag, <RDML MESSAGES> to determine the position in the message presentation standard page to display the LANSA messages. The line containing this tag will be repeated for each LANSA message in your application.

In your message presentation standard page, you can embed LANSA tags. The only restriction is that the line containing the <RDML MESSAGES> tag must not contain any other LANSA tags.

When you run your LANSA Web enabled application, LANSA for the Web will determine if there are any messages to display in your application. If there are messages, it will check if there is a message presentation standard page. If such a page exists, this page will be used to define the format of the message presentation. If it does not exist, the default message presentation format will be used.

# 12.5 Using <RDML MERGE="&END">

This tag allows the LWEB\_JOB jobs to return to the pool of free jobs immediately.

Under normal behavior, all LANSA jobs time out, whether they are the traditional procedural style or WEBEVENT functions. When the jobs time out, the user gets a 'Job Timed Out' message for the traditional procedural RDML functions.

For WEBEVENT functions, LANSA for the Web manages the data exchange (especially browse list data) and restarting of a new job automatically on subsequent interactions. The user does not see a 'Job Timed Out' message.

For WEBEVENT functions, LANSA terminates as soon as a DISPLAY or REQUEST command is processed. However, the LWEB\_JOB is still reserved for the user in case the user wants to interact further. The WEBEVENT job is not returned to the pool of free jobs immediately, not until the job is timed out (using the time out setting on your site).

The <RDML MERGE="&END"> tag is used with applications with high transaction rates. Typically, these are functions that display search results. Once the results are displayed, no further interaction with the function is expected from the user.

LANSA for the Web does not save any data for these jobs. This means that no browse list data is saved. If you include this tag in your HTML, DO NOT expect browse list data or any of the HTTP environment variables (e.g. \*WEBUSER) to be exchanged between this function and any subsequent function calls.

In summary, this tag should only be used for functions where you're absolutely sure that there is no further interaction with the function or you're not relying on the HTTP environment variables or the browse list does not need to be exchanged.

# **12.6 Cascading Style Sheets**

LANSA for the Web allows you to use cascading style sheets (CSS) with your Web function applications. Cascading style sheets allow you to describe a style that applies across one or more of your HTML pages. You can use CSS to define the presentation of an HTML page, including the font (face, size and color), background color, the positioning of elements within your HTML page as well as text decoration such as italics or underline.

For more details on cascading style sheets, refer elsewhere for information on Dynamic HTML (DHTML).

LANSA for the Web does not provide you with a CSS. You can create a CSS for your applications by creating a standard page named as DEFAULT\_STYLE. In this page, you can define the presentation attributes of the elements in your applications.

The DEFAULT\_STYLE page allows you to embed the style information directly into the HTML pages generated for your application. The CSS definitions are encapsulated by the <style> and </style> HTML tags.

If you have Web enabled your LANSA applications prior to Release 7.5 H5, you will need to recompile and regenerate the HTML for your functions.

Alternatively, you can modify the HTML pages manually and insert the <RDML MERGE="&STYLE"> statement just after the line containing the <title> element in the <head> section.

```
<RDML MERGE="&DTD_TRANSITIONAL">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title><RDML MERGE="&FUNCTION"></title>
<RDML MERGE="&STYLE">
</head>
```

## **Example of Using a Cascading Style Sheet**

FONTPREF1, FONTPREF2 and FONTPREF3 are created as Text Web components. These Web components contain the font families designated for the elements in your HTML page. The browser will attempt to locate and use the fonts in the order they are specified in the Web component.

```
For example, FONTPREF1 can be:
```

```
"Lucida Handwriting", "Arial", "Times New Roman"
```

To enable the CSS support, you will need to create the DEFAULT\_STYLE page. An example of this page is provided below:

0	
<style td="" ty<=""><td>ype="text/css"></td></tr><tr><td>body</td><td>{ font-family: <RDML COMPONENT="FONTPREF1">;</td></tr><tr><td>-</td><td>font-size: smallest;</td></tr><tr><td></td><td>color: gray;</td></tr><tr><td></td><td>margin-left: 5% }</td></tr><tr><td>table</td><td>{ font-family: <RDML COMPONENT="FONTPREF2">;</td></tr><tr><td></td><td>font-size: smallest;</td></tr><tr><td></td><td>color: gray }</td></tr><tr><td>h1, h2, l</td><td>h3 { font-family: <RDML COMPONENT="FONTPREF1">;</td></tr><tr><td></td><td>font-size: large;</td></tr><tr><td></td><td>color: darkblue }</td></tr><tr><td>strong</td><td>{ font-family: <RDML COMPONENT="FONTPREF1">;</td></tr><tr><td></td><td>color: gray;</td></tr><tr><td></td><td>font-size: small }</td></tr><tr><td>input</td><td>{ font-family: <RDML COMPONENT="FONTPREF3">;</td></tr><tr><td></td><td>color: gray;</td></tr><tr><td></td><td><pre>font-size: smaller }</pre></td></tr><tr><td>select</td><td>{ font-family: <RDML COMPONENT="FONTPREF2">;</td></tr><tr><td></td><td><pre>font-size: smallest }</pre></td></tr><tr><td></style>	

The CCS1 specifications require the font names to be included in quotes. Some browsers may work without the quotes, but regardless, font names should be included in quotes.

## 12.7 Retrieve Additional Information from Browse List

LANSA for the Web allows you to retrieve additional information from a record in a browse list. Typically, the information is stored in fields which have been set up to be hidden in the browse list. In addition, these fields contain information which are normally not required in the display.

By having these fields hidden, you can maximize the usage of the browser's display area to display the browse list. You can then set up a field in the browse list as a hot spot, i.e. a field which has a hyperlink to retrieve the additional information.

When you select any of the hyperlinked fields, a separate browser window (like the Help window) is used to display the additional information.

This feature will involve RDML programming. You will have to set the value of a field programmatically, following the convention:

```
<a href="javascript:GetExtraInfo('<list name>',
'<entry number>',
'<field #1>',
'<field #2>',
'<field #3>',
'<field #4>',
'<field #4>',
'<field #5>')"><text></a>
where
```

list name> is the name of your browse list (DEF\_LIST name),

<entry number> is the record number in the browse list,

```
<field #1> to <field #5> are the names of the fields you want to retrieve information from,
```

<text> could be some text you set or the contents of another field concatenated to this working field.

An example of the call would be:

<a href="javascript:GetExtraInfo('ListName', '001', 'field1', 'field2', 'field3', '&a The JavaScript function, GetExtraInfo, is provided for you by LANSA for the Web. It is defined in the DEFAULT\_SCRIPT page.

If you want to use this feature, you must observe the following:

- Your browser must be enabled for JavaScript support.
- Note the double quote (") characters used for the JavaScript function call.
- Note the single quote (') used to delimit each parameter passed to the JavaScript function, GetExtraInfo.
- You can specify up to a maximum of 5 fields to retrieve the information. Field parameters that are not used must be initialized to '&NULL'. (See the sample HTML above).
- You must not enable the Allow selection from any column in table option in the LANSA for the Web Administrator.

## 12.8 Handle the ENTER key in Browsers

LANSA for the Web uses the onsubmit JavaScript method to detect when the user has submitted the Form. The onsubmit JavaScript method calls a JavaScript Function HandleENTERKey to check if the PROCESS and FUNCTION Hidden Fields are Blank. If the fields are blank - an alert message is issued and the form is NOT submitted as the HandleENTERKey Function returns False. In WEBEVENT Functions, you can set the PROCESS and FUNCTION Hidden fields to merge in the current process using the <RDML MERGE="\*PROCESS"> and the current Function using the <RDML MERGE="\*FUNCTION"> tags if the WEBEVENT Function is calling the same Function to continue processing. The HandleENTERKey JavaScript as well as the FORM Tag with the onsubmit method is shown below:

```
<form name="LANSA" method="post" action="CGI-BIN/LANSAWEB?
WEBEVENT+L0192D93983F9389E0293+WEB" onsubmit="return HandleE]
```

```
function HandleENTERKey()
{
    if (document.LANSA._PROCESS.value == " " " ||
        document.LANSA._FUNCTION.value == " ")
    {
        alert("Form cannot be submitted using the ENTER button. Click on a BUT
        return true;
    }
    else
    {
        return false;
    }
}
```

# 12.9 Embed a Calendar Control

LANSA for the Web provides you with a calendar control that can be embedded into your application.

S	M	Т	W	Т	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

This calendar control can enhance the presentation of your application if it requires a calendar. If you need this calendar control, you will need to modify the STDHEADER page, either the default or a process specific page.

You will want to include an extra image into the toolbar. When the image is selected, it shows the calendar control. When you select a date from the calendar control, it populates the field with the selected date.

You will need to include the following line into your STDHEADER page:

<a href="javascript:CallCalendar()">

<img src="/IMAGES/TB\_CAL.GIF" alt="Calendar" border="0" width="70" h </a>

The calendar shipped with LANSA for the Web is configured to return the date in a DD/MM/YY format.

In order to the use this JavaScript function (CallCalendar( )), you must

remove the conditional RDML tags (ONCONDITION) from the calendar JavaScript in the DEFAULT\_SCRIPT.

# 12.10 Modify LANSA for the Web Messages

LANSA for the Web displays messages to the user under various circumstances. For example, data validation message are displayed when records are inserted into a file.

If you need to modify any of these messages, edit the LWEB.DAT file in the LANSA shared library.

- Do not modify the sequencing of the lines within the file.
- Do not modify the JavaScript functions.
- Do not add or delete any lines in the file.
- Do not modify any of the words prefixed with the '%' character.

# **Multilingual Modifications**

If you are using a non-English system and if you are encountering errors when you select the 'Home' key or messages are not displayed correctly, you may need to modify the syntax of the JavaScript functions.

You need to ensure that the following lines in the LWEB.DAT file look like this: Note, in the following lines:

- (1) represents the Primary Extension Name
- (2) represents the Secondary Extension Name
- (3) represents the Line Id.
  - (1) (2) Template (3) Text Data

Group

Name

- HTM ERRORPAGE 0 <head><title>Error</title>
- HTM ERRORPAGE 1 <script type="text/javascript" language="javascript"> //<![CDATA[
- HTM ERRORPAGE 2 function HomePage(){ parent.location="%s"; }
  //]]>
- HTM ERRORPAGE 3 </script></head><body><hr /><br /><h1>%s</h1><br /><hr /><br /><br /><br /><br /><br />
- HTM ERRORPAGE 4 <a href="javascript:HomePage()">Return to Home Page</a></body></html>
- HTM HOMEPAGE 0 <head><title>Display Home Page</title>
- HTM HOMEPAGE 1 <script type="text/javascript" language="javascript"> //<![CDATA[

- HTM HOMEPAGE 2 function HomePage(){ parent.location="%s"; } //]]>
- HTM HOMEPAGE 3 </script></head><body onload="HomePage()"> </body></html>
- HTM PREAMBLE 0 <!-- <?xml version="1.0"?> -->
- HTM PREAMBLE 1 <! DOCTYPE html PUBLIC "-
- //W3C//DTD XHTML 1.0

Transitional//EN"

- HTM PREAMBLE 2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1transitional.dtd">
- HTM PREAMBLE 3 <html xmlns="http://www.w3.org/1999/xhtml">
- HTM RESETPAREN 0 <head><title>Reset Parent</title>
- HTM RESETPAREN 1 <script type="text/javascript" language="javascript">//<![CDATA[
- HTM RESETPAREN 2 function Reset(){ parent.location= "%s://%s:%d/%s/%s?RESET+%s"; } //]]>
- HTM RESETPAREN 3 </script></head><body onload="Reset()"> </body></html>

In particular, you need to ensure that the '!' (Exclamation mark) character is correct, as well as the '{' and '}' (braces) characters.

## 12.11 Set the Initial Focus in an HTML Page

LANSA for the Web provides you with a facility to run a JavaScript function once the HTML page has been loaded by your browser.

By default, LANSA for the Web generates a LANSA tag that embeds a Web component as part of the <body> tag.

```
<body bgcolor="<RDML MERGE="*LW3CLNTCOLOR">" background="
```

```
<RDML MERGE="*LW3CLNTBKGND">" <RDML COMPONENT="FOR"
```

This component is known as FORMINIT. This component does not exist in the LANSA for the Web component registry, by default. However, this is a facility which can be used to execute a JavaScript function as part of the form loading function by the browser.

To activate this facility, create the FORMINIT Web component as a Text Web component. Define the text to be:

```
onload="SetFocus()"
```

Once this FORMINIT Web component is defined, it means that for every HTML page, it will attempt to execute the SetFocus function as part of the form loading routine by the browser.

```
<body bgcolor="white"
background="/IMAGES/BACKGRD.GIF" onload="SetFocus()">
```

The JavaScript function, SetFocus, can be incorporated into your DEFAULT\_SCRIPT page. It traverses through all the elements defined in your HTML form, looking for the first instance of an input field or a text area. Once it has found any of these objects, it sets the initial focus to that form element.

```
function SetFocus()
{
  var NumElements=document.LANSA.elements.length;
  for (i=0; i<NumElements;i++)
   {
    if (document.LANSA.elements[i].type=="select-one" ||
        document.LANSA.elements[i].type=="checkbox" ||
        document.LANSA.elements[i].type=="textarea" ||
</pre>
```

```
document.LANSA.elements[i].type=='text')
{
    if (document.LANSA.elements[i].value!="")
        document.LANSA.elements[i].select();
        document.LANSA.elements[i].focus();
        break;
    }
}
```

# 12.12 Tailoring the DEFAULT\_SCRIPT

The standard DEFAULT\_SCRIPT shipped with LANSA for the Web is extensive and contains some functions which are not essential for all applications. Consequently it is possible to tailor the script to contain only the functions required by your application. This will result in improved execution performance because the volume of data being transferred to the browser is reduced.

A tailored script can be created for a specific process and named

<process name>\_SCRIPT

This script will be used when the specific process is executed.

If you wish to save your tailored script as DEFAULT\_SCRIPT, it is strongly recommended that you first save a back up version of the full script.

Script function	Required/ Optional	When required
SetButton	Optional	This function must be included if you are using non-scrolling headers. Refer to Non Scrolling Header.
HandleSubmit	Optional	This function must be included if you are using non-scrolling headers. Refer to Non Scrolling Header.
PathOnly	Required	
GetHelp	Optional	This function handles a request for Help, so it must be included if your functions have Help enabled.
GetExtraInfo	Optional	This function is required if you are retrieving additional information from a Browse List. Refer to 12.7 Retrieve Additional Information from Browse List.
SetSelect	Required	

# SetNameLocation Required

CallCalendar	Optional	This function handles the Calendar control and must be included if you have used this feature. Refer to 12.9 Embed a Calendar Control.
SetDate	Optional	This function handles the Calendar control and must be included if you have used this feature. Refer to 12.9 Embed a Calendar Control
HandleEvent	Optional	This function is required for WEBEVENT functions.
IsValidNumeric	Optional	This function is required if the Validate numerics option on the Process/Function compile was set to YES.
IsDigit	Optional	This function is required if the Validate numerics option on the Process/Function compile was set to YES.
IsValidDBCS	Optional	This is required if the LANSA development language is DBCS.
IsDBCSChar	Optional	This is required if the LANSA development language is DBCS.

# **12.13 Generate Static Page Output to the IFS (Integrated File System)**

The WEB\_STATIC\_PAGE Built-in Function allows you to output Static HTML Pages directly to the IFS (Integrated File System). When you use this Built-in Function, all DISPLAY / REQUEST statements in the RDML will be sent to the IFS. Using this BIF to produce static HTML Pages allows users to access the Static HTML, rather than dynamically generating the page each time a function is run from the Browser. Static Pages can also be served up quicker than running a LANSA function each time a user request is made from the Browser.

This Built-in Function can only be used in a Web-enabled process.

When using this BIF, the LANSA System owner or object owner must have sufficient authority to create, delete and write to a file in the specified directory or Default Path on the IFS - they will not be created. It is recommended that DEFAULT\_SCRIPT, STDHEADER, STDFOOTER, MESSAGES and STYLESHEETS for the Process containing the Function be customized to be suitable for a static page, thus reducing the size of the generated HTML Page. A status message indicating the success or failure will be issued. Error Codes and Descriptions in the status message can be found in IBM's standard file SYS in library QSYSINC member ERRNO.

This Built-in Function can only be used in a Web-enabled process and can only be run in BATCH mode. This is possible because there is no screen interaction when using this BIF.

LANSA Functions that make use of the WEB\_STATIC\_PAGE BIF can be executed via the Web Browser or a non-programmable terminal (NPT or green screen). When the BIF is used and the function is run from the Browser, all HTML will be sent to the IFS based on the parameters (e.g. Filename, Suffix, Full Output Path on the IFS and Code page used to generate the IFS File) that have been passed into the BIF. If no parameters are passed, default parameters will be used. Refer to WEB\_STATIC\_PAGE in the LANSA Technical Reference Guide for more information.

In a multi-tier deployment, IFS output will be sent to the Web server (front-end) only when the function is executed from the Browser. If the Function is executed from a non-programmable terminal in a multi-tier

scenario, IFS output will be sent to the Data Server (the back-end). Functions using this BIF can also be submitted to run in Batch to generate or update static

HTML Pages.

This example of the WEB\_STATIC\_PAGE BIF will output Employee Details to a static page on the IFS. Default Parameters will create the IFS File with the Process/Function Name as the Default Filename and Panel Id as the Default Suffix in the root directory of the IFS.(/).

```
OPTIONS(*DIRECT)
 FUNCTION
 ******
 GROUP BY NAME(#FLDLST) FIELDS((#EMPNO) (#SURNAME)
                                                              (#
      (#ADDRESS1) (#ADDRESS2) (#POSTCODE) (#PHONE)
                                                          (#FAX)
 ******** The WEB_STATIC_PAGE BIF used here will send all
 ******************* It will be setup with Default Parameters.
         BUILTIN(WEB STATIC PAGE)
 USE
 ******** Fetch Employee Details from Employee File for
 ******** the Employee No exchanged to his function.
          FIELDS((*ALL)) FROM FILE(PSLMST) WITH KEY(A1234)
 FETCH
 ********* If Employee exists, output to IFS else send MESSAGE
 IF_STATUS IS_NOT(*OKAY)
 MESSAGE MSGTXT('Employee not Found in PSLMST File')
 ENDIF
 ******** Output Display to IFS
           FIELDS((#FLDLST))
 DISPLAY
When the WEB_STATIC_PAGE BIF is used, various parameters can be used to
setup the output path, filename and suffix parameters.
If the BIF is used as follows, the IFS File will be created in
/DIRECTORY1/REPORT1/ with filename IFSFILE.htm.
         BUILTIN WEB STATIC PAGE WITH ARGS("'IFSFILE"' "'*NO'
 USE
      "/DIRECTORY1/REPORT1/"")
```

If the BIF was setup as follows, the IFS File will be created in the root directory (/) with filename IFSFILE001.htm – the Default Suffix is the PanelID.

USE BUILTIN WEB\_STATIC\_PAGE WITH\_ARGS("'IFSFILE''' "\*DFT"' "'/"')

## **12.14 Integrate LANSA Applications with Static HTML Pages**

You can embed a LANSA application as part of your static pages or you can embed a static page into your LANSA application. This technique uses the Server Side Include (SSI) facilities provided by your Web serving product.

For more details on SSI support, refer to the Web serving product manuals.

If you want to embed a LANSA application as part of your static page, you will need to enable SSI support for that static page. In your static page, you will include a SSI instruction to execute the LANSA application. In this scenario, the HTML generated by the LANSA application will be part of the final HTML sent to the browser, incorporating the static information from the static page.

If you are only using SSI in a number of your HTML pages, it is more expedient to disable SSI Support at the system level (i.e. SSI Support is disabled in your system definition) and enable the SSI support at the HTML page level by using the <RDML SSI> tag.

If you want to enable an HTML page for SSI, make sure that the <RDML SSI> tag is at the beginning of the page.

```
<RDML SSI="ON">

<RDML COOKIES="&UDCOOKIES">

<!-- Process : PSLSYS Personnel System Main Menu -->

<!-- Function : ENROL Enrol a New Employee -->

<!-- Page : 001 -->
```

An example of a SSI instruction to launch a LANSA application is:

```
<!--#exec cgi="CGI-BIN/LANSAWEB?procfun+products+prodcat+web"-->
```

Your Web Server product may not support parameter passing in the URL syntax. In this case, you will need to create a CL program which calls the LANSAWEB program.

For example, your CL program may be:

#### CALL PGM(WWWCGI/LANSAWEB) PARM(PROCFUN PRODUCTS PRODCAT WEB)

If you want to embed a static page as part of your LANSA application, you can also use the SSI feature. In this case, you will modify the HTML generated for

your application to include the SSI instruction to include the static page. You must also enable the SSI support for your LANSA system. An example of a SSI instruction to embed a static page is:

<!--#include virtual="prodinfo.htm"-->

# 12.15 CheckBox Visual Web Components

You can use checkboxes to enhance the presentation of your applications. However, there is a limitation in using checkboxes in the interaction between the browser and the Web server. This is a limitation in the interaction between the browser and the Web Server. It is not a limitation imposed by LANSA for the Web.

In HTML syntax, you can only define a value for the ON (selected) state for the checkbox element. The browser only sends back a value for the checkbox if it is selected. This means that if the checkbox is not selected, no value is sent back by the browser.

You can visualize a field as a checkbox, by using the following HTML:

<input type="checkbox" name="RFLD" value="Y" />

This technique must only be deployed if the initial value of the checkbox is unchecked.

# Setting Initial State to Checked

If you visualize a field as a checkbox by just using the above technique, you will have problems if the initial value of the field is to set the checkbox ON (selected). If the user unchecks the checkbox, no value is sent back by the browser. This means that your application will not know that the field has been unchecked.

You can overcome this problem with a combination of a 'dummy' field and a JavaScript function.

This example should not be used to visualize fields in browse lists. This technique will not work in browse lists.

Your checkbox Visual Web component would contain the following:

```
<input type="hidden" name="RFLD" value="<RDML MERGE="RFLD">"
size="1" />
<input type="checkbox" name="DUMMY" onclick="SetState(this, 'RFLD', 'Y
<script type="text/javascript" language="javascript">
//<![CDATA[
if (document.LANSA.RFLD.value=="Y")
document.LANSA.DUMMY.checked=true;
//]]>
```

</script>

In the above example, RFLD is the actual name of the field you want to visualize as a checkbox. Instead of visualizing this field as a checkbox, it is hidden in the HTML. A 'dummy' field is used to visualize the checkbox. When the user clicks on the checkbox, a JavaScript function, SetState, is called to set the value of the actual field, RFLD.

The initial value of RFLD is read as a result of the <RDML MERGE="RFLD"> LANSA tag. The JavaScript function will set the state of the checkbox (the DUMMY field), dependent on the initial value of RFLD.

In this example, you assume that a value of 'Y' is the checked state. When the checkbox is unchecked the value is 'N'.

Note that the HTML for the Web component is not encapsulated by the <RDML CHECKVALUE> and </RDML> LANSA tags. You rely on the JavaScript function attached to the Web component to set the initial state of the checkbox.

The SetState JavaScript function is used by the DUMMY field to set the value of the actual field, RFLD. The third parameter of this function is the value for the checked state whilst the fourth parameter is for the unchecked state.

The SetState JavaScript function can be embedded into your DEFAULT\_SCRIPT page:

```
function SetState(obj, RFld, chkValue, unchkValue)
{
    if (obj.checked)
        RFld.value=chkValue;
    else
        RFld.value=unchkValue;
}
```

When the HTML form is submitted, the value attached to the RFLD field is returned to your application. The value attached to the DUMMY field is irrelevant since it is not used in your RDML function.

You will need to modify the check and unchecked values in the function according to your application.

# **Using Checkboxes in Browse list**

The above technique cannot be used for fields in browse lists because the names of the fields in browse lists are changed by LANSA for the Web dynamically. If you want to use checkboxes in browse lists with their initial set checked, you will need to deploy the following technique. This technique is based on the preceding example.

```
<input type="hidden" name="__{field name}-
<RDML MERGE="&ROWNUM" FORMAT=4> D"
value="<RDML MERGE="{field name}" size="{size}">" />
<input type="checkbox" name="DUMMY" value="<RDML MERGE="{field
name}">"
onclick="SetCBState(this, '__{field name}-
<RDML MERGE="&ROWNUM"
FORMAT=4> D', 'Y', 'N')"
<RDML ONCONDITION="{field name}">
checked="checked"
</RDML>
/>
```

{field name} is the name of the field you have defined in your DEF\_LIST command. This {field name} is padded with trailing blanks to 10 characters.

Note that this technique uses the <RDML ONCONDITION> tag to determine the initial state of the checkbox.

This technique calls a different JavaScript function, SetCBState:

```
function SetCBState(obj, RFld, CY, CN)
{
var NumElements=document.LANSA.elements.length;
for (i=0; i<NumElements;i++)
{
    if (document.LANSA.elements[i].name==RFld)
    {
        if (obj.checked) document.LANSA.elements[i].value=CY
        else document.LANSA.elements[i].value=CN;
        break;
    }
}</pre>
```

# 12.16 Extend LANSA Drop Downs

LANSA for the Web provides you with two choices with visualizing your fields as drop downs. You can either visualize the field as a Visual Web component or the field can be defined to be a LANSA drop down in your LANSA Repository.

If your field is defined to be a LANSA drop down in your repository, LANSA for the Web will automatically visualize the field as a drop down. LANSA drop downs are useful when the data to populate the drop down is dynamic. Visual Web components are useful when the data is fairly static in nature.

When you compile your function, the LANSA drop down fields are identified as <RDML MERGE="&DD<name>" FIELD="<field>"> in the generated HTML.

LANSA for the Web allows you to extend the drop down in a number of ways. You can choose to:

- Specify an offset position for the display of the drop down data.
- Specify that you wish to apportion the drop down data into VALUE and DESCRIPTION parts. The VALUE part is returned to your program while the DESCRIPTION part is displayed to the user.
- Visualize the drop down as a list box.

The syntax to extend the LANSA drop down is:

```
<RDML MERGE="&DD<name>" FIELD="<field>" OFFSET=
<position>" SPLIT
SIZE=<size>>
```

By default, you are unable to apportion the drop down data into its VALUE and description parts. For example, the drop down data was "ADMAdministration", this will be displayed in the drop down. The same value is also returned to your program.

LANSA for the Web allows you to apportion the value of your drop down data. In the example above, you may want "ADM" to be returned as the VALUE to your program, but only display "Administration" in the drop down. You can achieve this by editing the RDML tag and extending the tag.

The OFFSET keyword instructs LANSA for the Web to start from the offset position when displaying the drop down data. In the example above, you may only want to show "Administration" in the drop down, without the department code, "ADM". In this case, the offset position would be 4.

If you only extend the RDML tag by specifying the OFFSET keyword, the value returned to your application would be the drop down data. In the example,

this would be "ADMAdministration".

You can only use the SPLIT keyword if you specify an offset. If the SPLIT keyword is specified, the value returned to your application will be the VALUE portion of the apportioned data. In the example above, if the SPLIT keyword were specified, "ADM" is the value returned to your application. "Administration" will be the portion displayed to the user.

If you specify the SIZE keyword, you can convert the drop down into a list box. The size attribute determines the size of your list box. For example, if you specify SIZE=4, the LANSA drop down field will be visualized as a list box, with the size of the list box set to be 4 entries deep.

## 12.17 Modifying charset for non-English Systems

Some HTML components shipped with LANSA include a meta tag reference to charset 8859\_1. Depending on your system this may need to be modified to a charset appropriate for you installation.

For example:

```
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
```

These are the web components requiring modification in each partition in which they are installed:

DEFAULT\_LAYOUT

WIZLAYO01

WIZLAYO02

WIZLAYO03

WIZLAYO04

WIZLAYO05

**Note:** You will only have components WIZLAYO01-WIZLAYO05 if you have installed the e-Business Framework Wizard.

**WARNING:** Re-installing LANSA for the Web and e-Business Framework Wizard will re-set the charset to 8859\_1.

Refer to IANA Encoding in the LANSA Integrator Guide for a list of codes.

# **Tutorials for Web Functions & WEBEVENTs**

# What are the Web Function Tutorials?

The LANSA Web Function Tutorials are a set of exercises designed to introduce and reinforce the fundamental skills required to build Web Function Applications with LANSA. The tutorials are integrated into the online documentation.

To install the tutorials, refer to Tutorial Installation.

The following tutorials are included:

WEB001 - Types of LANSA Web Functions

Web Functions Wizard Tutorials

WEB002 - Coding a WEBEVENT Functions

WEB003 - iSeries Batch Debug

WEB004 - LANSA Generated HTML Pages

WEB005 - LANSA Process Pages

WEB006 - Graphic Variables

WEB007 - LANSA Tags

WEB008 - Web Components

- WEB009 Web Page Substitution (Optional Advanced)
- WEB010 Web Skeletons (iSeries)
- WEB011 Using DEFAULT\_HIDDEN
- WEB012 Dynamic Components

WEB013 - JavaScript and Browse Lists

- WEB014 Browse Lists
- WEB015 Data Apportionment

WEB016 - Customizing Personnel Application (Optional)

# Who Should Use the Tutorials?

The Tutorials can be used by novice and experienced LANSA developers who wish to learn how to build Web function applications with LANSA for the Web. In order to use the tutorials, you must be able to create LANSA processes and functions using templates. You should know how to edit RDML and how to

create fields in the LANSA Repository.

# Your Feedback

Your feedback regarding these tutorials will help us improve the overall quality of the LANSA documentation and training. Please email your comments to lansatraining@LANSA.com.au.

# How Do I Use the Tutorials?

It is recommended that you complete the tutorials in sequence. The processes and functions created in the first tutorials are reused in later tutorials.

The first steps in an exercise provide very precise descriptions of the tasks to be performed. As the steps and tutorials progress, the instructions become much more general.

All the tutorials use HTML. Be sure to use HTML mode when you use the Web Function Editor.

The tutorials use the files from the Personnel Demonstration System.

# What Partition Should I Use?

It is recommended that you use the DEM partition for the tutorial. The DEM is automatically installed with a LANSA for iSeries system and is quickly set up using Partition Initialization on a Visual LANSA system. The DEM system contains the Personnel System demonstration system which contains the files used by the tutorial.

If you do not use the DEM partition, you can set up another partition with the Personnel System files. Remember, the partition must be properly **Web**-enabled.

If you do not use DEM, you can select any of your partitions when using the tutorials. Testing or training partitions are ideal. You will need to import the Personnel System demonstration files into the partition.

If you intend to set up another partition, it is recommended that you do NOT use WEB as the partition identifier as this a reserved space and any changes to this partition may affect other Web-enabled partitions in your system.

# How Many Developers Can Use the Training?

There is no limit on the number of developers who may use the training at the same time. However, it is important that developers have a unique identifier for their work.

In the tutorial, each developer will use an object prefix iii which can be based on his or her initials or could be assigned by a system coordinator. For example, you will be asked to create a process named iiiPROC01. If your initials are JDS, you would create a process named JDSPROC01. This iii prefix must be unique in the LANSA system (not just the partition) as the iii prefix is used to create system variables. System variables are shared by all partitions.

# Setup Checklist

Check that these tasks have been completed before using the tutorials. If not, go to the instructions following and complete them as applicable:

- Have you **Web** enabled the partition?
- Do you have the Personnel System demonstration files (DEPTAB, SECTAB, PSLMST, etc.) installed into the partition?
- Have you loaded the graphic files used in tutorial WEB013? (See item 5 in the iSeries Installation Steps below.)
- Have you installed the Web Functions Wizard into the partition?

You will require this information about your specific installation:

- The site address for your **Web Server**
- The partition being used
- A user profile and password with authority to the LANSA partition
- A user profile and password if process authentication has been set up.

# **Tutorial Installation**

If you have not been able to answer "yes" the Setup Checklists question, you will need to carry out some or all of the following steps.

#### **iSeries Installation Steps**

- 1.Select the partition to be used for training. (You may wish to create a new partition.)
- 2.Use the PERSYS import on the LANSA iSeries Software CD-ROM to import the Personnel System application into the partition.
- 3. Web enable the partition. For details, refer to the Task: Set up IBM i Partition for Web Development in the *Installing LANSA on IBM i Guide*.
- 4. Use the LWEBWIZ2 import in the LANSA program library to import the Web Functions Wizard into the partition.
- 5. This step is optional.

For the results as shown for Tutorial WEB013, ten graphic files are required. (These files will be supplied if you do the LANSA Web training in a

classroom.)

You can create your own files, preferably images sized 144 x 155 pixels (2.000 x 2.083 inches). Name the images emp1.jpg through to emp10.jpg and load them to the LANSA default directory /images on your Web Server. You should also specify the file names in the Business Phone field of accessible records in the PSLMST file.

If these files are not in the /images directory, or not specified in the PSLMST file, it will not affect the objective of the exercise. Instead of an employee picture, you will simply see a box with an X, as no image is found.

# Windows Installation Steps

If you have a DEM partition with the Personnel System files already installed in your Visual LANSA system, then you should be albe to skip this step.

### **Create New Partition**

If you wish to create a brand new partition:

- 1.Follow the instructions to create a new Partition in Create New Partition in the *Visual LANSA User Guide*.
- 2.Initialize the new partition as described below.

# **Initialize a Partition**

To prepare your new or existing partition for use with these Tutorials, you will need to web enable the partition and install the Personnel System files. To do this, complete the following steps:

1.Log off Visual LANSA if you are logged on.

2.Log on to Visual LANSA.

NSA Logon		×
PCXUSER	DB User ID:	DBA
*****	Change <u>D</u> B Logon	
Current Language	Task ID	
English	PCXTASK	
tem Init	Messages Cancel	Help
	Current Language	PCXUSER     DB User ID:       xxxxxxx     Change DB Logon       Current Language     Task ID       English     PCXTASK

- 3.Highlight the partition to be used for training. If you don't have a partition to use, then create it as described above.
- 4.Enter your password, but do not press OK.

5. Press the Partition Init... button to open the Partition Initialization dialog box.

Partition Initialization	×
Mandatory Partition Initialization	
Visual LANSA Framework	
Enable for the Web	
🔽 e-Business Framework Wizard	
LANSA Client field and file definitions	
Personnel System Demonstration material (including executable objects)	
🗖 Run Demonstration	
OK Show Last Log Messages Cancel Help	

- 5.In the *Partition Initialization* dialog, select (that is, tick) the appropriate options. These are:
- *Enable LANSA for the Web*, if the partition is not yet web enabled.
- Web Functions Wizard.
- *Personnel System Demonstration material*, to obtain the data needed for the tutorials.

# WEB001 - Types of LANSA Web Functions

### **Objective:**

- To create a procedural function using a template and execute the process menu and function over the Web.
- To highlight how the transaction monitor supports procedural functions.
- To create a WEBEVENT function using a template and execute this function over the Web.
- To highlight the differences between procedural and WEBEVENT functions.

To achieve these objectives, you will complete the following steps:

- Step 1. Create a Procedural Function
- Step 2. Create a WEBEVENT Function
- Step 3. Execute your Procedural Function
- Step 4. Execute Your WEBEVENT Function
- Summary

# **Before You Begin**

You may wish to review these topics and all of their related sections:

- Introduction to LANSA Web Functions
- Developing Applications with LANSA Web Functions

In order to complete the tutorials, you must have completed the installation as described in the installation guide for the platform on which you are working. Refer to Personnel Demonstration System for more details about the sample files used in the tutorials. You will require the following information about your specific installation:

- the site address for your Web server
- partition being used
- a user profile and password with authority to the LANSA partition
- if process authentication has been set up, you will require a user profile and password.

### **Step 1. Create a Procedural Function**

In this step, you will create a test process and a procedural test function. The test function will use the FRENQ02 template to create a header/details style inquiry function using the Department (DEPTAB) and Section (SECTAB) files.

- 1. Using the LANSA development environment, signon to the partition nominated for the tutorials (usually DEM).
- 2. Create a new LANSA process named iiiPROC00 Test Process (Procedural), where iii are your initials. (If the process already exists, select a different set of characters for iii.)
- 3. Enable your iiiPROC00 process for web. If you need to know how, refer to Web Enabling a LANSA Process.
- 4. Working with your iiiPROC00 process, create a new function named iiiFN0 Display Sections Procedural. Be sure to specify that the function is generated from an Application Template.
- 5. Select the FRENQ02Header/Detail Inquiry template and answer the questions as shown below:

<b>TEMPLATE QUESTION</b>	ANSWER
Enter the name of the base file to be used by this template	DEPTAB
Select related files to be used	Select Section code table SECTAB
Do you want this function to be part of an action-bar style process?	Ν
Fields in Header Area	Select all fields
Fields in Detail/List Area	Select <b>DEPTMENT,</b> SECTION, SECDESC, SECPHBUS
Design the fields in the header are DOWN the screen or ACROSS the screen?	DOWN

- 6. You do not need to edit any of the RDML in this function.
- 7. Compile your iiiFN0 function. For more details, refer to Compiling Functions.

# **Step 2. Create a WEBEVENT Function**

In this step, you will create a test process and a test WEBEVENT function. The test function will use the FRWEBENQ01 template to create a WEBEVENT function which can be used to display information in the Department (DEPTAB) and Section (SECTAB) files.

- 1. Create a new LANSA process named iiiPROC01 Test Process (WEBEVENT), where iii are your initials. (If the process already exists, select a different set of characters for iii.)
- 2. Enable your **iiiProc01** process for web. If you need to know how, refer to Web Enabling a LANSA Process.
- 3. Working with your iiiPROC01 process, create a new function named iiiFN1 Display Sections WEBEVENT. Be sure to specify that the function is generated from an Application Template.
- 4. Select the FRWEBENQ01 template and answer the questions as shown below:

<b>TEMPLATE QUESTION</b>	ANSWER
Enter the name of the file to be used	SECTAB
Select keys to use for search	Select <b>DEPTMENT</b>
Fields to appear in browse list	Select <b>DEPTMENT, SECTION,</b> <b>SECDESC</b>
How many entries per page	5
Fields to appear in detail	Select all fields
What is the unique 2 character prefix	iii (where iii are your initials)

- 5. You do not need to edit any of the RDML in this function.
- 6. Compile your iiiFN1 function.

# **Step 3. Execute your Procedural Function**

In this step, you will execute your LANSA process menu and then execute your procedural function. You will see the frames used with the process menu. As you execute the procedural function, it will help you understand how the LANSA for the Web transaction server operates.

- 1. Check that your iiiFN0 function compiled successfully.
- 2. Open a browser window (Internet Explorer 5.0 or greater is recommended). To execute your process menu, enter this URL:

```
http://<server domain name>/cgi-bin/lansaweb?process+iiiPROC00+<ppp>
```

where:

<server domain name> is the domain name or IP address of your Web server

<ppp> is the LANSA partition

iii are your initials.

3. The process menu will appear something like this:

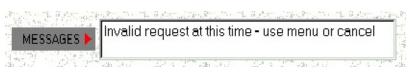


4. Using the process menu in the left frame, click on the Display Sections (Procedural) function to execute it.

5. Enter a Department code of ADM and press Next to see all Sections in the Administration department. Your function might appear something like the following:

						er				
le <u>⊑</u> dit ⊻	(iew Г <u>а</u> ур	rites <u>I</u> o	ols <u>II</u> el	Þ						
de 🕹	→> ÷ nward	Stop f	👔 Fefresh	A Horre	Q Search	Favorites	🎯 History	Mal	- Dint	
tress http	s://101.102.	103.104/0	IGI-BIN/L	ANSAWE	B?FUNCT	ON+L0219	06005066	820048F3	021 💌	∂Go
	ENU 🕨 NE	×t 🕨 e	ACK	SEARCH						
ebartuen	t Code	Å	DM	ectio ISTRAT			iceo	ura	9	
	t Code t Destrij	Å	DM DMIN	ISTRAT		PT Fhone	vceo		0	
epartmen Section	t Code t Destrij	,, А ціон, А Secti Descrip	DM DMIN DMIN	istrat F	TON Di Jusiness	PT Fhone ber	vcea		0	
epartment Section Code 01	t Code . t Destrij	, A tion . A Secti Descrip AL AD	DM DMIN DMIN MIN S	ISTRAT E RV 5	IION D Susiness Num	PT i Fhone ber	)CeO		0	
epartment Section Code 01	t Code t Destrij t NTERN	A tion . A Secti Descrip TAL AD ASING	DM DMIN on otion MIN S SECTI	ISTRAT E RV 6 ON 9	FION DI Business Num 79 2536	PT Phone ber	vcea		0	
Section Code 01 02	t Code t Descrip INTERN FURCIL	, A Secti Descrip AL AD ASING NTING	DM DMIN on otion MDN S SECTI	ISTRAT E RV 6 ON 9 ION 5	EION DI Business Num 79 2536 52 5475 60 3633	PT i Phone ber	vcea		Ð	

- 6. Press the *Next* button to return to the REQUEST for a Department Code. This time enter a Department code of FLT and press Next to see all Sections in the Fleet department
- 7. In order to demonstrate how the transaction server works, press the browser's Back button to return to the REQUEST for a Department Code.
- 8. Enter a Department code of AUD and press the Next button. You should receive the following type of error message:



The LANSA for the Web Transaction Server knows that the RDML function is waiting for an input from the DISPLAY statement. To navigate in procedural functions, you must use the buttons in the Web function application. For more details, refer to Example of a Procedural Function.

9. Use the *Menu* button to return to the process menu.

#### **Step 4. Execute Your WEBEVENT Function**

In this step, you will execute your WEBEVENT function to help you understand how it differs from the procedural function.

1. WEBEVENT functions are always called directly. To execute your WEBEVENT function, enter this URL:

http://<server domain name>/cgi-bin/lansaweb?procfun+iiPR0C01+iiFN001+<ppp>

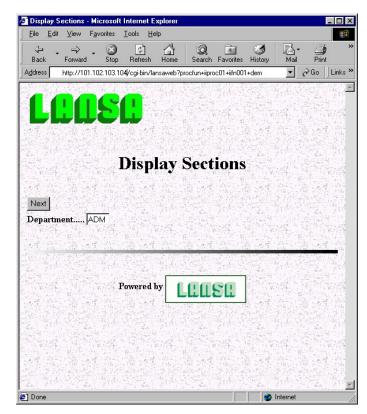
where:

<server domain name> is the domain name or IP address of your Web server

<ppp> is the LANSA partition

iii are your initials.

Your function might appear like this:



2. Enter a Department code of ADM and press Next to see all Sections in the Administration department. Your function might appear like this:

<u>F</u> ile <u>E</u> dit <u>V</u> iew	F <u>a</u> vorites <u>T</u> oc	ols <u>H</u> elp						1923
↔ → Back Forward	🚽 🙆 Stop R	😰 🚮 lefresh Home	e Search	* Favorites	3 History	Mail	<b>Print</b>	
Address 🙋 /CGI-BIN	I/LANSAWEB?w	/EBEVENT+L02	219040D5586	800021B306	64+DEM+E	NG	∂Go	Links
La								
	D	isplay	Sect	ions				
AT 18-14-14-1	的目标后用			5 TA (				1 - 1 - 1 
Next Page	New Search	1				1.5 2 3 5		
	New Search	a share of the	and the second	常情感	San S	對國際的	and the second	14
Dept Sectio	n S	Section escription						
Dept Sectio	n S e De	Section	SRV					
Dept Sectio Code Code	n S De De INTERNA	Section escription	-36 . N					
Dept Sectio Code Code O ADM 01	n S De INTERNA PURCHAS	Section escription L ADMIN S	ION					
Dept Section Code Code C ADM 01 C ADM 02	n S De INTERNA PURCHAS ACCOUN	Section escription L ADMIN S SING SECT	ION TION					
Dept Sectio Code Code C ADM 01 C ADM 02 C ADM 03	n S De INTERNA PURCHAS ACCOUN	Section escription L ADMIN S SING SECT TING SECT TING SECT	ION TION					
Dept Sectio Code Code C ADM 01 C ADM 02 C ADM 03 C ADM 04	n S De De INTERNA PURCHAS ACCOUN SALES AN	Section escription L ADMIN S SING SECT TING SECT TING SECT	ION TION					
Dept Sectio Code Code C ADM 01 C ADM 02 C ADM 03 C ADM 04	n S De De INTERNA PURCHAS ACCOUN SALES AN	Section escription L ADMIN S SING SECT TING SECT TING SECT	ION TION					
Dept Sectio Code Code C ADM 01 C ADM 02 C ADM 03 C ADM 04	n S De De INTERNA PURCHAS ACCOUNT SALES AN MAINTEN	Section escription L ADMIN S SING SECT TING SECT ID MARKE IANCE	ION TION TTING					
Dept Sectio Code Code C ADM 01 C ADM 02 C ADM 03 C ADM 04	n S De De INTERNA PURCHAS ACCOUNT SALES AN MAINTEN	Section escription L ADMIN S SING SECT TING SECT ID MARKE IANCE	ION TION	50				

- 3. Use the browser's Back button to return to the previous screen.
- 4. Enter a Department code of FLT and press Next to see all Sections in the Fleet department.

Notice that no error message is displayed. The WEBEVENT function fully supports the browser's Back button. For more details, refer to Example of a WEBEVENT Function.

#### Summary

#### **Important Observations**

- Procedural functions do not support the use of the browser's Back button. In order to go back in the procedural paradigm, the user must use an equivalent to the F12=Cancel key or button.
- You must always call your WEBEVENT functions directly. You can not call them from a process menu.
- When testing your Web function applications, remember to use one of the buttons (Next, Search, Menu, Ok, etc.) on the page to submit the data back to the server. Do not press the Enter key unless specifically instructed. If you wish to have the Enter key submit your page, you must add a JavaScript function to handle this event. Refer to Handle the ENTER key in Browsers.

### **Tips & Techniques**

- The majority of your Web function applications will be built using WEBEVENT functions.
- If you are building applications for an Intranet, you may wish to Web enable your existing LANSA applications instead of rebuilding WEBEVENT applications.

# What I Should Know

- How to Web enable a LANSA process and its functions.
- How to execute a LANSA process using a standard browser.
- How to directly execute a LANSA function using a standard browser.
- How procedural functions are different from WEBEVENT functions when they execute over the Web.

# **Other Tutorials**

You may wish to proceed to WEB002 - Coding a WEBEVENT Functions.

### **Web Functions Wizard Tutorials**

Depending on your final Web requirements, you could do the Web Functions Wizard Tutorials now or you could do them all at the end of this workshop. The Web Functions Wizard Tutorials are in the *Web Functions Wizard Guide*.

#### **Objective:**

To learn how to use the Web Functions Wizard to customize your LANSA Web function applications. When you do the Wizard tutorials you will learn how to:

- Customize browse lists and process specific layouts.
- Use the Wizard to define menu components.
- Adopt presentation layouts.
- Customize a Presentation Layout
- Customize a Menu Component
- Customize System Wide Attributes
- Import/Export an Application

# WEB002 - Coding a WEBEVENT Functions

#### **Objectives:**

- To understand how to create your own WEBEVENT functions by rewriting a procedural Header/Detail style function as a WEBEVENT function.
- To understand how WEBEVENT RDML logic differs from procedural logic.
- To learn how to link WEBEVENT functions.
- To learn how to create re-entrant WEBEVENT functions.

To achieve these objectives, you will complete the following steps:

- Step 1. Review Procedural Logic
- Step 2. Create New Functions
- Step 3. Define Keywords for Function Routing
- Step 4. Test Your WEBEVENT Functions
- Step 5. Re-entrant WEBEVENT Function
- Summary

# **Before You Begin**

You may wish to review these topics and all of their related sections:

• WEBEVENT Functions

In particular, you should review the following:

• WEBEVENT Example

In order to complete the tutorials, you should have completed the following:

• WEB001 - Types of LANSA Web Functions

# **Step 1. Review Procedural Logic**

In this step, you will review the RDML logic of procedural functions and you will redesign the function as two WEBEVENT functions. Each WEBEVENT function will have just one display statement. For more details, refer to How Is WEBEVENT Different?

1. Following is a sample of the code which is produced by the FRENQ02 template using the DEPTAB and SECTAB files:

```
FUNCTION OPTIONS(*NOMESSAGES *DEFERWRITE *DIRECT)
GROUP_BY NAME(#HEADER) FIELDS(#DEPTMENT #DEPTDESC)
DEF LIST
NAME(#iiiLIST) FIELDS((#LISTDUMMY *HIDDEN) #SECTION #SECDE
******** COMMENT(Loop until user EXITs or CANCELs)
BEGIN LOOP
R10: REQUEST
FIELDS(#DEPTMENT) DESIGN(*DOWN) IDENTIFY(*DESC)
********* COMMENT(Fetch file DEPTAB details
                                          )
FETCH
FIELDS(#HEADER) FROM_FILE(DEPTAB) WITH_KEY(#DEPTMENT) N
********* COMMENT(Select all file SECTAB details)
SELECT
FIELDS(#iiiLIST) FROM_FILE(SECTAB) WITH_KEY(#DEPTMENT)
ADD_ENTRY TO_LIST(#iiiLIST)
ENDSELECT
```

**\*\*\*\*\*\*** COMMENT(Display results to the user)

DISPLAY

CHANGE FIELD(#HEADER) TO(\*DEFAULT)

CLR\_LIST NAMED(#iiiLIST)

END\_LOOP

Notes: You will use a list name of #iiiLIST, where iii=your initials.

2. Redesign this function using two WEBEVNT functions. For an example, refer to How Does WEBEVENT Work? (Solutions are provided in Step 2.)

## **Step 2. Create New Functions**

In this step, you will create two new functions with the code you have designed from the previous step.

- 1. Create a new LANSA process named iiiPROC03 WEBEVENT Functions, where iii are your initials. (If the process already exists, select a different set of characters for iii.)
- 2. Enable your iiiPROC03 process for web. If you need to know how, refer to Web Enabling a LANSA Process.
- 3. Working with your iiiPROC03 process, create a new function named iiiFN5 Request Department for Search. You will manually enter the code for the function.

The RDML code in the function might appear something like this:

FUNCTION OPTIONS(\*DIRECT \*WEBEVENT) CHANGE FIELD(#DEPTMENT) TO(\*DEFAULT) REQUEST FIELDS(#DEPTMENT) DESIGN(\*DOWN) IDENTIFY(\*DESC) MENU\_KE

4. Working with your iiiPROC03 process, create a new function named iiiFN6 Display Department/Section. You will manually enter the code for the function. (TIP: You could start this new function by copying code from your iiiFN0 function.)

The RDML code in the function might appear something like this:

FUNCTIONOPTIONS(\*DIRECT \*WEBEVENT)GROUP\_BYNAME(#HEADER) FIELDS(#DEPTMENT #DEPTDESC)DEF\_LIST

NAME(#iiiLIST) FIELDS((#LISTDUMMY \*HIDDEN) #SECTION #SECDE CLR\_LIST NAMED(#iiiLIST)

\*\*\*\*\*\*\*\*\*\* COMMENT(Fetch file DEPTAB details)

FETCH

FIELDS(#HEADER) FROM\_FILE(DEPTAB) WITH\_KEY(#DEPTMENT) N \*\*\*\*\*\*\*\*\* COMMENT(Select the SECTAB file details)

SELECT

FIELDS(#iiiLIST) FROM\_FILE(SECTAB) WITH\_KEY(#DEPTMENT) ADD\_ENTRY TO\_LIST(#iiiLIST)

ENDSELECT

**\*\*\*\*\*\*\*** COMMENT(Display results to the user)

R10: DISPLAY FIELDS(#HEADER) DESIGN(\*DOWN) IDENTIFY(\*DESC) BROWSELIS<sup>\*</sup>

5. Compile the new functions.

## **Step 3. Define Keywords for Function Routing**

In this step, you will use the Web Function Editor to define your WEBEVENT function routing.

1. Start the Web Function Editor. You will be asked to enter these details:

System	
Host Type	
Partition	
Language	
User Profile	
Password	
Mode	HTML

Be sure to specify HTML mode.

- 2. Use the Tools menu category and select the Keywords Maintain option. Do not enter a Process. Press the *OK* button to continue.
- 3. Press the *Add* button to create a link from the request function to the display function when the user performs a search. Enter the following information:

Process	iiiPROC03
Function	iiiFN5
Keyword	SEARCH
Description	Search
Linked Process	iiiPROC03
Linked Function	iiiFN6

4. Create another link from the display back to the request so another search can be performed. Enter the following information:

Process	iiiPROC03
Function	iiiFN6
Keyword	RETURN
Description	New Search
Linked Process	iiiPROC03
Linked Function	iiiFN5

**Reminder:** Keyword is case sensitive! It must match the keyword used in your RDML.

For more information, you can use the online help in the Web Function Editor.

5. Close the Maintain Keywords window.

#### **Step 4. Test Your WEBEVENT Functions**

- 1. Check that your functions compiled successfully.
- 2. Open a browser window (Internet Explorer 5.0 or greater is recommended) and execute your iiiFN5 function as follows:

#### http://<server domain name>/cgi-bin/lansaweb? procfun+iiiPROC03+iiiFN5+<ppp>

where:

<server domain name> is the domain name or IP address of your Web server

<ppp> is the LANSA partition

iii are your initials

- 3. Test your new WEBEVENT function as follows:
  - Enter a Department of ADM.
  - Use the Search button to display a list of Sections in the ADM department.
  - Use the browser's Back button. Notice that the Department Code is ADM. Enter a Department of FLT and repeat the search.
  - Try using the New Search button to perform another search. Notice that the Department Code is now blank when the REQUEST panel is displayed.
- 4. Try executing your iiiFN6 WEBEVENT Display Department/Section function directly from the browser.

Notice what happens. Why do you receive the error message that no record could be found? For the answer, refer to the Important Observations in the Summary.

# **Step 5. Re-entrant WEBEVENT Function**

In this step, you will create a single re-entrant WEBEVENT function to perform the same operations as the iiiFN5 and iiiFN6 functions.

- 1. Working with your iiiPROC03 process, create a new function named iiiFN7 Display Sections in Department. You will manually enter the code for the function.
- 2. Write the RDML code as a re-entrant WEBEVENT function so that it requests a Department Code and then displays the Sections in the Department. For more details, refer to Handling Re-entrant Functions.
- 3. Compile your function.
- 4. Remember to register the keywords for the function. In this case, the function links back to itself.
- 5. Test your function.

# Solution

This is one possible solution to this exercise:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
DEFINE
FIELD(#RENTRY) TYPE(*CHAR) LENGTH(1) DEFAULT(*BLANK)
GROUP BY
NAME(#HEADER) FIELDS((#DEPTMENT) (#DEPTDESC) (#RENTRY *H
DEF LIST
NAME(#iiiLIST) FIELDS((#LISTDUMMY *HIDDEN) #SECTION #SECDE
     COND('#RENTRY *NE Y')
IF
         FIELD(#HEADER) TO(*DEFAULT)
CHANGE
         FIELD(#RENTRY) TO(Y)
CHANGE
REQUEST
FIELDS((#DEPTMENT) (#RENTRY *HIDDEN)) DESIGN(*DOWN) IDEN
ELSE
CLR LIST NAMED(#iiiLIST)
********* COMMENT(Fetch file DEPTAB details )
FETCH
FIELDS(#HEADER) FROM_FILE(DEPTAB) WITH_KEY(#DEPTMENT) N
********** COMMENT(Select the SECTAB file details)
SELECT
FIELDS(#iiiLIST) FROM FILE(SECTAB) WITH KEY(#DEPTMENT)
```

```
ADD_ENTRY TO_LIST(#iiiLIST)
ENDSELECT
******** COMMENT(Display results to the user)
R10: CHANGE FIELD(#RENTRY) TO(*BLANK)
DISPLAY
FIELDS(#HEADER) DESIGN(*DOWN) IDENTIFY(*DESC) BROWSELIST
RETURN))
ENDIF
```

The keyword entries are:

Process	iiiPROC03
Function	iiiFN7
Keyword	SEARCH
Description	Search
Linked Process	iiiPROC03
Linked Function	iiiFN7

Process	iiiPROC03
Function	iiiFN7
Keyword	RETURN
Description	New Search
Linked Process	iiiPROC03
Linked Function	iiiFN7

#### Summary

#### **Important Observations**

- The Data Exchange is automatically handled by LANSA for the Web. You did not have to code exchange of data between iiiFN5 and iiiFN6. When you execute iiiFN6 directly, you receive an error message because no data was passed to the function. iiiFN6 usually receives its data (i.e. the Department Code) from iiiFN05.
- The #RENTRY field must be included as a \*HIDDEN variable in the REQUEST statement so that it is passed along with the #DEPTMENT when the iiiFN7 function is called to DISPLAY the search results.
- When you compile your re-entrant WEBEVENT function, you may have received compile warnings about the structure of your function. For more details, refer to Considerations for WEBEVENT Functions.
- If you have completed the Web Functions Wizard tutorials, you will notice that the browse list presentation for your iiiLIST defaults to the layout you defined with Wizard.

### **Tips & Techniques**

- You can use Web Link components instead of keywords. For an example, refer to WEB008 Web Components.
- You can use the Web Functions Wizard to enhance the layout of your function.

#### What I Should Know

- How to convert a procedural function into WEBEVENT function(s).
- How to write a re-entrant WEBEVENT function.
- How to use the Web Function Editor Keywords to link WEBEVENT functions.

# **Other Tutorials**

You are now ready to complete WEB004 - LANSA Generated HTML Pages.

# WEB003 - iSeries Batch Debug

#### **Objective:**

• To learn how to use batch debug with WEBEVENT functions on the iSeries.

To achieve this objective, you must complete the following:

- Step 1. Identify Your Terminal ID
- Step 2. Call Your Function in Debug Mode
- Step 3. Enter valid data into your browser.
- Summary

### **Step 1. Identify Your Terminal ID**

1. If you are using 5250 terminals or emulation sessions, signoff the session and look at the display name in the top right corner of the signon screen.

File Edit	n B - [24 × Transfer	(80) Appearance	Comn	nunicati	on	As <u>s</u> ist	Wind	ow	Help		_	
	<u> </u>	. PETER IN 199	1.000	Sign			11.14					
				01911			6ystem Gubsys† )isplaį	tem .			SYDASW QINTER QPADEV	
	F F M	lser 'assword . 'rogram/proc lenu Urrent libr	edure · · ·									
					(C)	COPY	/RIGHT	IBM	CORP.	1980,	1999.	
A b											0	6/0

2. Record the name of the device. Do not sign on to this device!

### **Step 2. Call Your Function in Debug Mode**

Before you start this step, make sure that the 5250 device is displaying a user login screen.

1. Using the browser, execute your process iiiPRO03 and function iiiFN07 by entering the URL for the LANSA function, and include the following:

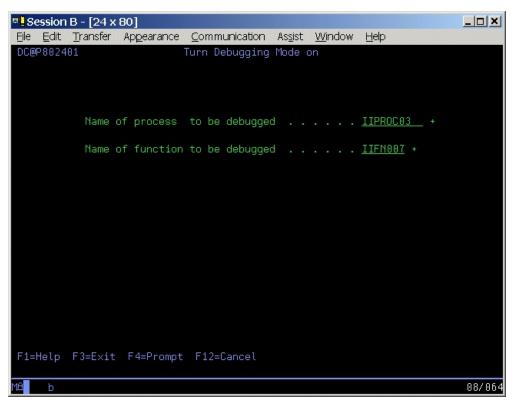
+BDEBUG+devicewhere device is the name you recorded in Step 1

For example: the URL might look something like this:

http://siteaddress/cgi-bin/LANSAWEB?

PROCFUN+iiiPROC03+iiiFN07+DEM+BDEBUG+QPADEV0001

2. Switch to the 5250 display you have nominated. You will see the standard LANSA debug interface.



3. Specify the iiiPROC03 process and iiiFN07 function for debug.

Press Enter to continue.

4. Your display screen shows three options. 1.DEBUG Interactively, 2.TRACE All/Selective statements, and 3.COUNT statements.

Select option 1.DEBUG Interactively:

200000000000000000000000000000000000000	B - [24 x 80] Transfer Appearance <u>C</u> ommunication As <u>s</u> ist <u>Wi</u> ndow <u>H</u> elj	p
C@P80240		•
	Name of process to be debugged <u>IIP</u>	<u>ROC03</u> +
	: F@DEBUG02 Display Sections in Department	
	: • Colort DEPUE option and process Entern	
	: Select DEBUG option and press Enter	
	: <u>1</u> 1. DEBUG Interactively	
	: 2. TRACE All/Selective statements	
	: 3. COUNT statements	
	: F3=Exit	
	•	
	definition IIFN007 from IIPROC03 now locked by this	s job
1=Help	F3=Exit F4=Prompt F12=Cancel	

5. A listing of your RDML source code for function iiiFN07 will appear. Beside each statement you will notice there is a selection box. You can select break points for individual lines of code or all lines of code.

For this exercise, use F20-Select all:

Session B -	[24 x 80]	
<u>Eile E</u> dit <u>T</u> ra	ansfer Ap <u>p</u> ear	rance <u>C</u> ommunication As <u>s</u> ist <u>W</u> indow <u>H</u> elp
F@DEBUG02		Display Sections in Department
		Mode : DIS
		debug mode. Choose from the list below the commands
		r to execution for the display of debug information.
Position at S	Seq	
Find		
Dbug Seq Lbl		
		Beginning of RDML commands *********
1	FUNCTION	OPTIONS(*DIRECT *WEBEVENT)
2	DEFINE	FIELD(#RENTRY) TYPE(*CHAR) LENGTH(1) DEFAULT(*BLANK)
.3	GROUP_BY	NAME(#HEADER) FIELDS((#IIDEPT) (#DEPTDESC) (#RENTRY *HI
		DDEN))
4	DEF_LIST	NAME(#IILIST) FIELDS((#LISTDUMMY *HIDDEN) (#SECTION) (#
		SECDESC) (#SECPHBUS))
_ 5	IF	COND('#RENTRY *NE Y')
_ 6	CHANGE	FIELD(#HEADER) TO(*DEFAULT)
_ 7	CHANGE	FIELD(#RENTRY) TO(Y)
_ 8	REQUEST	FIELDS((#IIDEPT) (#RENTRY *HIDDEN)) DESIGN(*DOWN) IDENT
		IFY(*DESC) EXIT_KEY(*NO) MENU_KEY(*NO) ADD_KEY(*NO) PRO
		MPT_KEY(*NO) USER_KEYS((1 SEARCH))
F16=F/Fwd	F17=F/Bkwd	F20=Select All
МАВЬ		15/003

6. A listing of all fields referenced by your function should now appear. Please select the fields, IO\$STS and RENTRY.

Press Enter to allow your function to begin execution.

Session B - [24	-				<u>_     ×</u>
	Appearance <u>C</u> ommunication		Help		
F@DEBUG02 Select Functio	Display Sections in Variables to be displayed.			de	: DIS
Search For Var:	able :				
ct Field name	RPG name Description			Tupe	Len Dec
DEPTDESC	@IR001 Department Desc	ription		A	20
_ IIDEPT				Ĥ	4
X IO\$STS	IO\$STS Data base I/O op	peration status	s code	Ĥ	2
LISTDUMMY	@IR002 List dummy cont	rol field		A	1
X RENTRY	RENTRY RENTRY			A	1
SECDESC	@IR003 Section Full Des	scription		Ĥ	20
_ SECPHBUS	@IR004 Business Phone	Number of Secti	ion	A	15
_ SECTION	@IR005 Section Code			Ĥ	2
F20=Select All					
Ъ					13/00

7. The first executed line of source code will appear with the variables IO\$STS and RENTRY shown below it.

Press Enter to advance the debugger through each line of code. Remember, for WEBEVENT functions, a screen is not sent to the browser until the function terminates.

Take notice of the value of RENTRY and when it changes from blank to Y.

8. Press Enter until the DEBUG screen resets to a signon display.

Now that your function has terminated, a screen should be displayed in the browser.

### Step 3. Enter valid data into your browser.

- 1. Switch to your browser.
- Enter a value of ADM in the search field and press the Next key to submit the function.

Your iiiFN07 will be executed once again.

2. Switch to the 5250 display screen you have nominated.

You should see the standard LANSA debug interface.

Simply press Enter. (The \*FIRST options will automatically take the first function in the debug process.)

- 3. Select Debug Interactively and press Enter.
- Your previous break points should be selected. If not, press F20 again.
   Your variables IO\$STS and RENTRY should already be selected. If they are not, then select them again.
- 5. Notice the value of RENTRY is still a Y. This value was set when the function executed the first time. The value was stored in the hidden values.

Press Enter to move to next break point in the function.

You will be able to watch the function as it selects the data from the file and prepares the output screen.

- 6. Press Enter until the function terminates and an iSeries signon is displayed.
- 7. Switch to your browser. You should see the list of sections displayed.

### Summary

### **Important Observations**

- The device used for batch debug must be active (i.e. the 5250 workstation must be powered on or the 5250 emulation session started). A user should not be signed on to the terminal. You cannot use batch debug if the display station is already allocated.
- The interactive and batch debug have identical features.
- The batch debug can be nominated either in the URL directly, or it can be called from your generated HTML pages.

### **Tips & Techniques**

- Remember, you must not be signed on to the device nominated for batch debug.
- WEBEVENT functions terminate immediately after the REQUEST or DISPLAY statement.
- You cannot debug a function that is locked by a developer. A common mistake is to have the function editor open while attempting to debug the function.

### What I Should Know

• How to use the batch debug with WEBEVENT functions.

How to set break points and how to display variables in your functions.

# WEB004 - LANSA Generated HTML Pages

### **Objective:**

- To execute the Web Function Editor and review some of the basic Editor features.
- To learn how to identify the LANSA generated HTML pages.
- To review the HTML pages created for the iiiFN05 Request Department function. The HTML will be manually edited and the function will be recompiled to show how versions and the Web Function Editor's Compare and Contrast features work.
- To become familiar with the default (page components) and HTML documents generated by LANSA.

The focus of this tutorial is NOT the HTML code generated or the RDML tags used by LANSA. You will review the use of LANSA tags in an upcoming exercise.

To achieve these objectives, you complete the following steps:

- Step 1. Open and Configure the Web Function Editor
- Step 2. Identify Generated HTML Pages
- Step 3. Edit the HTML Page for iiiFN05
- Step 4. Modify and Recompile Function iiiFN05
- Step 5. Use Compare and Contrast to Review your HTML
- Summary

# **Before You Begin**

You may wish to review these topics and all their related sections:

• LANSA Generated HTML/XML Pages

In order to complete the tutorials, you should have completed the following:

• WEB002 - Coding a WEBEVENT Functions

It is very important that you have configured LANSA for the Web to allow for automatic backup of the generated HTML pages. Use the LANSA for the Web Administrator to check these Data/Application Server settings.

### **Step 1. Open and Configure the Web Function Editor**

In this step, you will start the Web Function Editor and configure the Editor options.

1. Start the Web Function Editor. You will be asked to enter the following:

System
Host Type
Partition
Language
User Profile
Password

Mode HTML

Be sure to specify HTML mode.

- 2. Use the Options menu category and select the Configure option.
- 3. Select the *View* tab. Check the box for Synchronized Scrolling. Make sure the *Vertical Split* radio button is selected. These options are used when comparing versions of HTML pages.

Configuration Opti	ons		×
Third Party <u>E</u> ditor	⊻iew Misce	llaneous	
Com I	pare Background nserted Text	d Colors indows Split	
	ОК	Cancel	Help

4. Select the *Miscellaneous* tab. The option to *Enable archive functionality when saving* allows you to archive manual changes to the HTML function. Refer to Configure, Miscellaneous tab for further information.

Configuration Options	x
Third Party Editor View Miscellaneous	
Most Recently Used Files Options	
Items in MRU list: 4 (Range 0-10)	
Clear MRU List	
Max. Undo Actions: 50 (Range 1-32000)	
Enable archive functionality when saving	
OK Cancel Help	

5. Press the *OK* button to save the settings.

### Step 2. Identify Generated HTML Pages

In this step, you will simply identify the HTML pages created for function iiiFN07 which was compiled in WEB002 - Coding a WEBEVENT Functions. (Remember to substitute iii with your initials.)

- 1. Review the RDML code used in your iiiFN07 function. (Refer to Step 5. Reentrant WEBEVENT Function.) Count the number of REQUEST and DISPLAY statements in the function.
- 2. Use the File menu category and select the Open option. A list of HTML pages stored in the LANSA internal database will be displayed. Not all of these pages are generated from LANSA functions. Some pages are defaults shipped with LANSA. Some pages are Page Web components defined by developers.
- 3. Look for (but do not open) the iiiPROC03 iiiFN07 pages from your reentrant WEBEVENT function.

Because iiiFN07 has two panels (REQUEST and DISPLAY), you should see two files listed:

iiiPROC03 iiiFN07001

iiiPROC03 iiiFN07002

The first page will be for the REQUEST panel and the second will be for the DISPLAY panel. A single page is created for each display. For more details, refer to Identifying Generated Pages.

IIPROCO0 IIFN000001	Process IIPROCO0 FL			
IIPROCO0 IIFN000002	Process IIPROCO0 Fu			
IIPROCO1 IIFN001001	Process IIPROC01 Fu			
IIPROCO1 IIFN001002	Process IIPROC01 Fu			
IIPROCO1 IIFN001003	Process IIPROC01 Fu			
IIPROCO1 IIFN001004	Process IIPROC01 Fu			
IIPROC03 IIFN005001	Process IIPROC03 FL			
IIPROCO3 IIFN006001	Process IIPROC03 FL			
IIPROCO3 IIFN007001	Process IIPROC03 FL			
IIPROCO3 IIFN007002	Process IIPROC03 Fu			
Name:	Version: 0			
Language: ENG 💌				
Description:				
OK. Delete	Cancel			

### Step 3. Edit the HTML Page for iiiFN05

In this step, you will locate and open the HTML page of function iiiFN05 which was created in WEB002 - Coding a WEBEVENT Functions.

1. Locate and open the iiiPR0C03 iiiFN05001 page which will have the HTML for the REQUEST panel from your WEBEVENT function.

**Reminder:** The RDML code in your iiiFN05 function might appear something like this:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
CHANGE FIELD(#DEPTMENT) TO(*DEFAULT)
REQUEST
FIELDS(#DEPTMENT) DESIGN(*DOWN) IDENTIFY(*DESC) MENU_KE
```

- An HTML page will be downloaded from the LANSA Application/Data Server to your Web Function Editor. This is the LANSA internal HTML which includes LANSA tags, components, graphic variables, etc.
- 2. You will make two simple changes to this file. (These changes are being made to show how a new version of HTML will be created when you save the document, and when you recompile the function. The image file used in this step is shipped with LANSA.)

Insert the bolded lines which will simply add an image and some text to the page:

<RDML LAYOUT>

<pre><!-- Process : IIIPROC03 WEBEVENT Function <! Function : IIIFN005 Request Department </pre--></pre>	for Search>
	->
Generated by - LANSA</td <td>&gt;</td>	>
Created by user - xxxxxxxxxx</td <td>&gt;</td>	>
Time and Date - 000000000000</td <td>&gt;</td>	>
RDML function sequence number - 0003</td <td>&gt;</td>	>
This is a *WEBEVENT function</td <td>&gt;</td>	>
<form <="" method="post" name="LANSA" td=""><td></td></form>	

action="/<RDML MERGE="&CGI">/LANSAWEB?WEBEVENT+ <RDML MERGE="&SESSPL">">

<RDML MERGE="&HIDDEN">

<RDML COMPONENT="STDHEADER">

<center><h1><RDML MERGE="&FUNCTION"></h1></center>

<br />

```
<RDML MERGE="&MESSAGES">
```

<br />

```
<img src="/IMAGES/LANSA.GIF" alt="Logo" /><br/>
```

<basefont size="" />

<br/>br /> Enter a Department Code to see a listing of all Sections.<br/><br/>br />

<br />

#### <RDML MERGE="&BUTTONS">

### <RDML COMPONENT="STDFOOTER">

</form>

3. Save the file.

A message box will appear to ask you if you wish to archive the previous version of the HTML page. Press the *Yes* button so that the original version of the HTML page will be saved as Version 1. The current version is always Version 0.

4. Use your browser to execute function iiiFN05 to view your changes.

Notice that you have not recompiled your function.

### Step 4. Modify and Recompile Function iiiFN05

In this step, you will modify the RDML in function iiiFN05 and recompile the function so that a new version of the HTML page is created. The changes made in Step 3 will allow you to use the Compare and Contrast features in the Editor.

- 1. Working with the iiiFN05 function in process iiiPRCO03, use the LANSA RDML Editor to view the RDML code in the function.
- 2. Manually edit the RDML code to add the #STD\_DATE field to the REQUEST statement. (You will not use the #STD\_DATE field in this exercise. This step will simply highlight how your RDML changes are identified in the new HTML page.) Your RDML statement should appear as follows:

```
REQUEST
FIELDS(#DEPTMENT #STD_DATE) DESIGN(*DOWN) IDENTIFY(*DES(
```

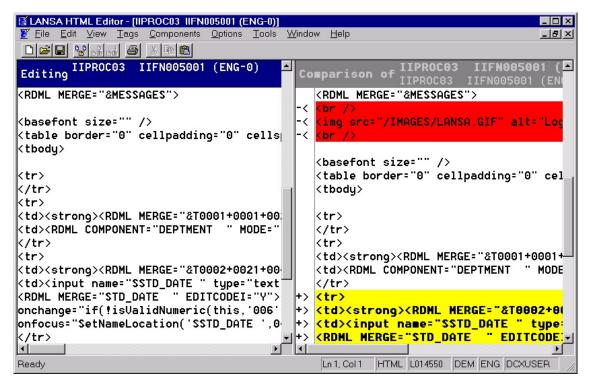
- 3. Save the changes to the function and exit the RDML Editor.
- 4. Submit your iiiFN05 function for recompile. This step will cause the HTML page to be recreated.

Reminder: The current page is always Version 0. The previous HTML page, which includes the manual changes you have just made, will be saved as Version 1. The very first page created will now be Version 2. For more details, refer to Versioning of Pages.

### Step 5. Use Compare and Contrast to Review your HTML

In this step, you will compare the HTML created when the function was recompiled, to the manually edited HTML.

- 1. Check that the function iiiFN05 has compiled successfully.
- 2. Using the Web Function Editor, open the iiiPR0C03 iiiFN05001 page.
- 3. Review the HTML. You will not see the manual HTML changes you made in Step 3 since the recompile creates a new version of the HTML. (Your changes have been saved as Version 1.)
- 4. Use the File action bar category and select the Compare With Version option. Select Page Version 1 and press *OK*. The version of the HTML with your changes will be displayed.
- 5. Scroll up and down through the HTML listing. The differences between the two files will be highlighted in red and yellow. (These colors can be configured in the editor.) For more details, refer to Comparing Versions.



- 6. Cut and paste the changes you made to your HTML in Step 3 to the current version of the file.
- 7. Save the document.
- 8. Finally, open Version 1 of the iiiPR0C03 iiiFN05001 page. Try to save the

document. Notice the error message displayed. Close the document without saving it.

### Summary

### **Important Observations**

- An HTML page is created for every REQUEST and DISPLAY statement in a function.
- The order of the REQUEST and DISPLAY statements determines the document number.
- Versions of HTML pages can be created when a document is saved. This feature is controlled by the Web Function Editor options.
- New versions of the HTML pages are created when a function is compiled. This feature is enabled with the LANSA for the Web Administrator.
- The current version of an HTML document is always Version 0.
- You can only save your changes to an HTML document as Version 0.

### **Tips & Techniques**

- If you do not wish to use the automated archive and backup features, you can backup your HTML by using the Save As option. You can manually save your documents using different document names.
- You can control the archiving and backup of the HTML pages from the Web Function Editor options and from the LANSA for the Web Administrator set up.
- For an example of how to protect custom changes to your HTML, refer to WEB009 Web Page Substitution (Optional Advanced).

### What I Should Know

- How to use the basic features of the Web Function Editor to view HTML pages.
- How to identify the LANSA generated HTML pages.
- How the Web Function Editor uses versions.
- When versions of HTML pages are created.

# **Other Tutorials**

You are now ready to complete WEB005 - LANSA Process Pages.

### WEB005 - LANSA Process Pages

### **Objective:**

- To highlight how process specific Web pages can be created.
- To edit the message presentation used in your LANSA functions.
- To create new standard headers and footers for your functions.
- To edit the JavaScript to add a calendar control to the function.
- To introduce the component registry.

To achieve these objectives, you will complete the following steps:

- Step 1. Create a Message Presentation Page for iiiPROC03
- Step 2. Create a Standard Header for Functions in Process iiiPROC03
- Step 3. Create a Standard Footer for Functions in Process iiiPROC03
- Step 4. Component Registry
- Step 5. Add the Calendar Control and Edit the Default JavaScript
- Step 6. Test the Calendar Control
- Summary

### **Before You Begin**

You may wish to review these topics and all of their related sections:

• Default Process Pages

In order to complete the tutorials, you must have completed the following:

- WEB002 Coding a WEBEVENT Functions
- WEB004 LANSA Generated HTML Pages

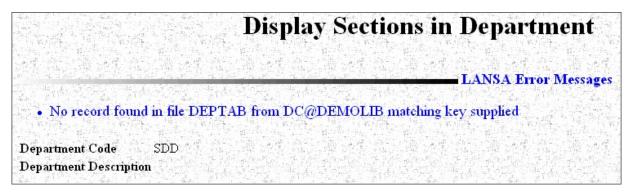
### Step 1. Create a Message Presentation Page for iiiPROC03

In this step, you will customize the presentation of the messages in a specific process to replace the default list box style messages. You will create a very simple list to display messages. (Remember to substitute iii with your initials.)

- 1. Using the Web Function Editor in HTML mode, create a new page. Use the File menu category and select the New option.
- 2. Delete the default HTML which appears in the new document.

```
Enter the following HTML:
<img src="/IMAGES/GRADIENT.GIF" alt="" />
<font size="4" color="blue">
<b>LANSA Error Messages</b>
<RDML MESSAGES>
</font>
```

- 3. Save the HTML document as iiiPROC03\_MSGPRES.
- 4. Test your changes using your iiiFN07 function. Try searching for a Department which does not exist. Notice that you did not have to recompile any functions or edit the function HTML. Your message should appear something like the following:



### **Step 2. Create a Standard Header for Functions in Process iiiPROC03**

In this step, you will create a new standard header page to be used specifically with the functions in process iiiPROC03. The new header will simply have some additional heading text.

- 1. Make note of the current appearance of the function header and footer in any of the functions in your iiiPROC03 process.
- 2. Using the Web Function Editor, open the STDHEADER.
- 3. Insert the bolded lines which will simply add a heading to the page:

```
<!--
                           -->
  <!--
        LANSA for the Web
                                   -->
  <!--
        Standard Header
                                  -->
  <!--
        Last Modified: 11.3 2006-05-21 -->
  <img src="<RDML MERGE="*LW3CPYLOGO">" alt="Logo" /><br />
<h4> iii LANSA Web Applications</h4><br />
  <RDML BUTTON="&WEBEVENT">
  <script type="text/javascript" language="javascript">
  //<![CDATA[
  function ButtonClick(button)
  {
   document.LANSA._BUTTON.value=button; document.LANSA.submit();
  }
  //]]>
  </script>
  </RDML>
  <table cellpadding="0" cellspacing="0" border="0" width="100%" align="left
  •••
```

4. Use the *Save As* option to save the document as:

### iiiPROC03\_STDHEADER.

### **Step 3. Create a Standard Footer for Functions in Process iiiPROC03**

In this step, you will create a new standard footer page to be used specifically with the functions in process iiiPROC03. The new footer will simply have an additional copyright statement.

- 1. Open the STDFOOTER document.
- 2. Insert the highlighted lines which will add some text to the page:

```
<!--
                        -->
<!--
     LANSA for the Web
                               -->
<!--
     Standard Footer
                              -->
     Last Modified: 11.3 2006-05-21 -->
<!--
<br />
<img src="</pre>
<RDML MERGE="*LW3IMGFBORDER">" alt="" />
<br />
<h4 align="center">Powered by <a href="http://www.lansa.com"><img src="
<RDML MERGE="*LW3IMGLANSA">" align="middle" border="0" alt="La
</a></h4>
```

# <br/> <br/> <h4 align="center">Copyright LANSA 2007.</h4>

3. Use the *Save As* option to save the document as:

### iiiPROC03\_STDFOOTER

4. Execute your iiiFN05 function. Notice that your changes to the header and footer do not appear. At this point, you have only created the HTML pages. The process specific component pages are not known to LANSA. You must register the new pages in the component registry before they can be used.

### **Step 4. Component Registry**

- 1. Using the Web Function Editor, use the Component menu category and select the Maintain option.
- 2. Press the *Add* button to define a new component.
- 3. Enter a component name of iiiPROC03\_STDHEADER and use the dropdown to select a Type of Page. Press *Continue*.
- 4. When the New Page Component dialog appears, enter the following:

Description	Standard Header
Page	iiiPROC03_STDHEADER
Mode	Not Applicable

- 5. Press *OK* to add the new component to the registry.
- 6. Repeat these steps to create a component named iiiPROC03\_STDFOOTER with the following information:

Description	Standard Footer			
Page	iiiPROC03_STDFOOTER			
Mode	Not Applicable			

7. Execute your iiiFN05 or iiiFN07 function (or any function in process iiiPROC03) and you will see the new header and footer. Notice that you did not have to recompile any functions or edit the function HTML.

# Step 5. Add the Calendar Control and Edit the Default JavaScript

In this step, you will enable and test the calendar control JavaScript function. You will also change the date format used in the JavaScript from MM/DD/YY to DD/MM/YY.

1. Open the iiiPR0C03 iiiFN05001 page and insert the bolded lines which will include a Calendar button beside the #STD\_DATE field.

```
<strong><RDML MERGE="@T0002+0021+0040"></strong>
>
```

```
<input name="SSTD_DATE " type="text" maxlength="008" align="right"
<RDML MERGE="STD_DATE " EDITCODEI="Y">
```

```
onfocus="SetNameLocation('SSTD_DATE ',04,29)">
```

```
<a href="javascript:CallCalendar()">
<img src="/IMAGES/TB_CAL.GIF" alt="Calendar" border="0" width=
</a>
```

- 2. Save the document.
- 3. Open the default JavaScript page called DEFAULT\_SCRIPT.
- 4. Remove the conditions on the JavaScript for the Calendar functions by deleting the two bolded lines in the following code:

```
<RDML ONCONDITION="USECALENDAR">
```

```
function CallCalendar()
{
    if (document.LANSA._CALFLD.value != "&NULL")
    {
        var opt = "width=340,height=385,directories=no,toolbar=no,"
        opt += "menubar=no,scrollbars=no,resizable=yes"
```

```
win = window.open("/images/lcalen.htm", "calendar", opt)
win.opener=window
}
function SetDate(day, month, year)
{
var Field=document.LANSA._CALFLD.value;
```

```
if (day < 10)
    day="0"+day;
if (month < 10)
    month="0"+month;
if (year < 2000)
    YearRet=year-1900;
else
    YearRet=year-2000;</pre>
```

```
if (YearRet < 10)
YearRet="0"+YearRet;
```

```
var RDate=day+"/"+month+"/"+YearRet;
```

var NumElements=document.LANSA.elements.length;

```
for (i=0; i<NumElements;i++)
{
    if (document.LANSA.elements[i].name==Field)
    {
        document.LANSA.elements[i].value=Rdate;
        break;
    }
}
</RDML>
```

5. Next, modify the script so that the calendar control returns the date in format

MM/DD/YY by changing the statement:

```
var RDate=day+"/"+month+"/"+YearRet;
```

to

var RDate=month+"/"+day+"/"+YearRet;

6. Use the Save As option to save the document as: iiiPROC03\_SCRIPT

### **Step 6. Test the Calendar Control**

- 1. Execute your iiiFN05 function.
- 2. Position the cursor in the date field.
- 3. Press the *Calendar* button.

ANSA Calendar - Microsoft Internet Explorer 📃 🗖							
December 🔽 2001 💌							
S	M	T	W	Τ	F	S	
						1	
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	
30	31						
	<b>S</b> 2 9 16 23	De <b>S M</b> 2 3 9 10 16 17 23 24	S         M         T           2         3         4           9         10         11           16         17         18           23         24         25	December         Image: mail of the state of the st	December       2001         S       M       T       W       T         2       3       4       5       6         9       10       11       12       13         16       17       18       19       20         23       24       25       26       27	December       2001         S       M       T       W       T       F         Q       Q       Q       Q       Q       Q       Q         Q       Q       Q       Q       Q       Q       Q       Q         Q	December         2001           S         M         T         W         T         F         S           I <tdi< td="">         I         <tdi< td=""></tdi<></tdi<>

4. Select a date from the calendar. The date should be returned in MM/DD/YY format.

### Summary

### **Important Observations**

- There is no DEFAULT\_MSGPRES page shipped with LANSA.
- The STDHEADER and STDFOOTER are Web Page components. These must be registered as components before they can be used.
- The DEFAULT\_SCRIPT is a process specific page. It does not have to be registered, as it is not a component.

### **Tips & Techniques**

- Creating process specific pages allows you to customize your functions in a specific process. You should not modify the DEFAULT pages unless the change applies to all functions in the partition.
- All process specific pages will be automatically exported with the process definition. These pages do not need to be registered as Web Page components.
- Conditional LANSA RDML tags can be used to control the amount of text being sent to the browser. The DEFAULT\_SCRIPT uses these tags only to send the required script functions.

# What I Should Know

- How to use process specific pages and page components.
- How to customize the message presentation for functions.
- How to use the default JavaScript and calendar control function.

# **Other Tutorials**

You are now ready to complete WEB006 - Graphic Variables.

### WEB006 - Graphic Variables

### **Objective:**

- To highlight how to create your own graphic variables in LANSA for the Web.
- Optional: To review the graphic variables created by the Web Functions Wizard.

If you are using the process specific or the browse list specific graphic variables, you can use the Web Functions Wizard to create and change these variables. You do not have to create these variables manually.

To achieve these objectives, you will complete the following steps:

- Step 1. Create New Graphic Variables
- Step 2. Add Graphic Variables to the Process Pages
- Step 3. Test Your Graphic Variable.
- Optional Step 4. Web Functions Wizard Graphic Variables
- Summary

### **Before You Begin**

You may wish to review these topics and all their related sections:

• Graphic Variables

In order to complete this tutorial, you must have completed the following:

- WEB002 Coding a WEBEVENT Functions
- WEB005 LANSA Process Pages

### **Step 1. Create New Graphic Variables**

- 1. Using the Web Function Editor, use the Component action bar category and choose the Graphic Variables option.
- 2. Press the *Add* button to define a new graphic variable called
  \*WEBiiiIMAGE (where iii is your initials). It should be of the type FILE. The image file name should be: LANSAANI.GIF

(The LANSAANI.GIF file is shipped with the LANSA software.)

- 3. Define a new graphic variable called \*WEBiiiCOLOR (where iii is your initials). It should be type COLOR. The text should be: RED
- 4. Define a new graphic variable called \*WEBiiiCOPYRIGHT (where iii is your initials). It should be type TEXT. The text should be: **Copyright LANSA 2006**.

### Step 2. Add Graphic Variables to the Process Pages

In this step, you will add the graphic variables to the process specific STDFOOTER page created in WEB005 - LANSA Process Pages.

1. Use the Web Function Editor to update the iiiPROC03\_STDFOOTER page. Make the following changes (in bold):

```
<!--
                         -->
 <!-- LANSA for the Web
                                 -->
 <!--
       Standard Footer
                               -->
 <!--
       Last Modified: 9.1 2001-09-10 -->
 <br />
 <img src="</pre>
 <RDML MERGE="*WEBiiiIMAGE">" alt="" />
 <br />
 <h4 align="center">Powered by <a href="http://www.lansa.com"><img src="
 <RDML MERGE="*LW3IMGLANSA">" align="middle" border="0" alt="L/
 </a></h4>
<br />
 <h4 align="center"><font size="5" color="
 <RDML MERGE="*WEBiiiCOLOR">">
 <RDML MERGE="*WEBiiiCOPYRIGHT"></font></h4>
```

2. Save the document.

### Step 3. Test Your Graphic Variable.

You will now test your graphic variables and demonstrate how they can be used to minimize application maintenance.

- 1. Execute function iiiFN05. Notice the new footer used in the function.
- 2. Use the Web Function Editor to change the name of the image file associated with the graphic variable \*WEBiiiIMAGE to: GRADIENT.GIF
- 3. Change the name of the color associated with the graphic variable \*WEBiiiCOLOR to #32CD32,which is limegreen.
- 4. Change the text associated with the graphic variable \*WEBiiiCOPYRIGHT to Copyright LANSA 2001.
- 5. Without recompiling or editing your HTML, execute function iiiFN05 again. Notice the changes to the image and text.

### **Optional Step 4. Web Functions Wizard Graphic Variables**

If you have completed the Web Functions Wizard Tutorials, you will have created a number of different graphic variables.

- 1. Using the Web Function Editor, use the Component action bar category and choose the Graphic Variables option.
- 2. Review the list of graphic variables and locate the following:

\*LW3BLACB\_iiiLIST2

\*LW3BLACF\_iiiLIST2

These are graphic variables created by the Web Functions Wizard for your browse list, iiiLIST2.

### Summary

### **Important Observations**

- The RDML MERGE tag is used to embed graphic variables.
- The color for a graphic variable can be specified using hexadecimal values (e.g. #32CD32) or using color names (e.g. limegreen).

### **Tips & Techniques**

- The text in a graphic variable can include HTML tags.
- Graphic variables should be used instead of hard coding values into your HTML pages.
- You should use the Web Functions Wizard for your process or browse list specific variables.
- If you wish to delete graphic variables, you must use the Web Function Editor. Graphic variables cannot be deleted from the Web Functions Wizard.
- Remember that graphic variables are defined at the system level. They are shared by all partitions.

# What I Should Know

• How to create file, image and text graphic variables.

# **Other Tutorials**

You are now ready to complete WEB007 - LANSA Tags.

### WEB007 - LANSA Tags

### **Objective:**

- To highlight how LANSA tags are used in the HTML pages.
- To manually add some LANSA tags to the HTML.
- To show how HTML can be controlled using LANSA tags.

To achieve these objectives, you will complete the following steps:

- Step 1. Review the LANSA Tags in iiiFN05
- Step 2. Add LANSA Tags to iiiFN05001 HTML Page
- Step 3. Modify Standard Process Footer
- Step 4. Test the function
- Summary

### **Before You Begin**

You may wish to review these topics and all of their related sections:

- What are LANSA Tags?
- How Do LANSA Tags Work?
- LANSA Tags Example
- Using <RDML> and </RDML> Tags
- <RDML BUTTON>
- <RDML MERGE>
- <RDML NOTCONDITION>

In order to complete this tutorial, you must have completed the following:

- WEB002 Coding a WEBEVENT Functions
- WEB004 LANSA Generated HTML Pages
- WEB005 LANSA Process Pages

### Step 1. Review the LANSA Tags in iiiFN05

- 1. Use the Web Function Editor to open the iiiPROC03iiiFN05001 page.
- 2. Notice how the RDML MERGE statement is used in the following examples:

<RDML MERGE="&HIDDEN"> <RDML COMPONENT="STDHEADER"> <center><h1><RDML MERGE="&FUNCTION"></h1></center> <br /> <RDML MERGE="&MESSAGES">

The MERGE can be used with special reserve words such as &HIDDEN for embedding hidden variables or &MESSAGES for embedding the LANSA messages. The MERGE of &FUNCTION is used for the function description.

3. Notice how the RDML MERGE statement is used in the following field example:

<input name="ADEPTMENT " type="text" size="004" maxlength="004" value="<RDML MERGE="DEPTMENT ">" onfocus="SetNameLocation('ADEPTMENT ',03,29)" />

The MERGE is used to embed the field multilingual label <RDML MERGE="&T0001+0001+0020">. It is also used to embed the field value <RDML MERGE="DEPTMENT ">.

4. The <RDML COMPONENT="STDHEADER"> and <RDML COMPONENT="STDFOOTER"> are used to embed the Web page components. Notice that these statements do not have to be modified to use the process specific pages, which were created in WEB005 - LANSA Process Pages.

# Step 2. Add LANSA Tags to iiiFN05001 HTML Page

In this step, you will use the RDML BUTTON and RDML ONCONDITION tags to show you how they control the presentation of the HTML.

1. Using the Web Function Editor to edit the iiiPROC03 iiiFN05001 page, add the following NOTCONDITION tag around the text shown. (This text was added in Tutorial 4. If it is not present, simply add all 3 lines to your page.) Your text should appear as follows:

```
<RDML NOTCONDITION="DEPTMENT">
Enter a Department Code to see a listing of all Sections.
</RDML>
```

The NOTCONDITION tag will check if the DEPTMENT is blank. When it is blank, it will use the HTML text within the NOTCONDITION tag.

2. Directly following this text, add an RDML BUTTON tag to display some text depending upon the buttons which are enabled. Your text should appear as follows:

```
<RDML BUTTON="&ADD">
This text will not be displayed since Add button is not enabled.
</RDML>
```

3. Save the document.

### **Step 3. Modify Standard Process Footer**

In this step, you will add the date to the bottom of the standard footer by using a LANSA system variable with the MERGE tag.

- 1. Open the iiiPROC03\_STDFOOTER document.
- Use a MERGE tag to include the LANSA system variables \*DAYC, \*MONTHC and \*YEARC. Add the following text to the end of the STDFOOTER document:

```
Date in DD/MM/YY format is: <RDML MERGE="*DAYC"> / <RDML M
</p>
```

3. Save the document.

#### Step 4. Test the function

1. Execute your iiiFN05 function. You should see the following text on the screen:

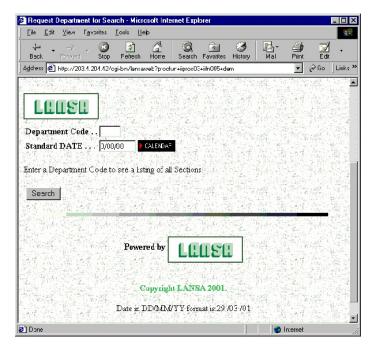
Enter a Department Code to see a listing of all Sections.

The following text should not appear.

This text will not be displayed since Add button is not enabled.

At the bottom of the screen, you should see today's date: Date in DD/MM/YY format is: 99/99/99

You function might appear something like the following:



#### Summary

## **Important Observations**

- The RDML tags can be used almost anywhere in your LANSA generated pages.
- The </RDML> tags must always be on a separate line with no other commands.
- The RDML MERGE command can be used with fields, graphic variables, and any LANSA system variable.

# **Tips & Techniques**

- You can use LANSA tags within your LANSA Web components.
- The NOTCONDITION and ONCONDITION tags are a very powerful method of controlling your HTML. For example, you can use these tags to limit the amount of text being sent to the browser. The DEFAULT\_SCRIPT uses these tags to only send the required script functions.

# What I Should Know

• How to use LANSA tags in your generated pages.

# **Other Tutorials**

You are now ready to complete WEB008 - Web Components

#### WEB008 - Web Components

#### **Objective:**

- To create some reusable Web components for your Web functions application.
- To show how components can be added manually and automatically in LANSA.
- To demonstrate how components can be used in other components.
- To introduce the concept of modes and Web components.

You have already used page components when creating the process specific STDHEADER and STDFOOTER in WEB005 - LANSA Process Pages.

To achieve these objectives you will complete the following steps:

- Step 1. Create a New Field in the Repository
- Step 2. Create Visual Web Component
- Step 3. Create a Text Web Component
- Step 4. Banner Web Component
- Step 5. Use Hidden Fields in Function
- Step 6. Create a Web Link Component
- Summary

# **Before You Begin**

You may wish to review these topics and all of their related sections:

- Introduction to Web Components
- Banner
- Text
- Web Link
- Page
- Naming Page and Script Web Components
- Visual Web Component
- Creating Visual Web Components

In order to complete this tutorial, you must complete the following:

- WEB002 Coding a WEBEVENT Functions
- WEB004 LANSA Generated HTML Pages

• WEB005 - LANSA Process Pages

# **Step 1. Create a New Field in the Repository**

To demonstrate how components are automatically embedded for a field, you must create a new field iiiDEPT (where iii are your initials) in the LANSA. The iiiDEPT field will be copied from the DEPTMENT field.

- 1. From the LANSA development environment, work with the fields in the LANSA Repository. Create a new field called iiiDEPT (where iii=your ID) by copying the DEPTMENT field which already exists in the LANSA Repository.
- 2. When you save the field, you may copy the help text, validation rules and multilingual definitions for the field.
- 3. Edit the RDML in your iiiFN07 function. Use the find and change utility in the Web Function Editor to change all occurrences of DEPTMENT to iiiDEPT. (There should be four changes.) Your RDML might appear as follows:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
DEFINE
FIELD(#RENTRY) TYPE(*CHAR) LENGTH(1) DEFAULT(*BLANK)
GROUP BY
NAME(#HEADER) FIELDS((#iiiDEPT) (#DEPTDESC) (#RENTRY *HIDD]
DEF LIST
NAME(#iiiLIST) FIELDS((#LISTDUMMY *HIDDEN) #SECTION #SECDE
     COND('#RENTRY *NE Y')
IF
          FIELD(#HEADER) TO(*DEFAULT)
CHANGE
          FIELD(#RENTRY) TO(Y)
CHANGE
REQUEST
FIELDS((#iiiDEPT) (#RENTRY *HIDDEN)) DESIGN(*DOWN) IDENTIFY
ELSE
CLR_LIST NAMED(#iiiLIST)
********* COMMENT(Fetch file DEPTAB details )
FETCH
FIELDS(#HEADER) FROM_FILE(DEPTAB) WITH_KEY(#iiiDEPT) NOT_
********** COMMENT(Select the SECTAB file details)
SELECT
FIELDS(#iiiLIST) FROM FILE(SECTAB) WITH KEY(#iiiDEPT)
ADD ENTRY TO LIST(#iiiLIST)
ENDSELECT
```

```
********* COMMENT(Display results to the user)
R10: CHANGE FIELD(#RENTRY) TO(*BLANK)
DISPLAY
FIELDS(#HEADER) DESIGN(*DOWN) IDENTIFY(*DESC) BROWSELIS<sup>-</sup>
RETURN))
ENDIF
```

- 4. Save the changes to the function and exit the editor.
- 5. Do NOT recompile your function. The function will be compiled once the new component has been created for iiiDEPT.

# Step 2. Create Visual Web Component

In this step, you will use the automated creation of Visual Web components to build a drop down list showing all departments in the DEPTAB file. The Visual Web component will be named iiiDEPT so that it will be automatically embedded into the HTML for the iiiFN07 function.

- 1. Start the Web Function Editor. Use the File menu category and select the Open option to view a list of HTML pages in the partition. Notice that there is no page with the name iiiDEPT.
- 2. Use the Components menu category and select the Generate Component Visual option.
- 3. Enter only the following information:

Component	iiiDEPT
Visual Type	Drop down
File	DEPTAB
Field for Value	DEPTMENT
Field for description	DEPTDESC

- 4. Press *OK*. LANSA for the Web will use the information in the DEPTAB file to build the HTML code for the drop down.
- 5. Recompile your iiiFN07 function.
- 6. Use the File menu category and select the Open option to view a list of HTML pages in the partition. Open the iiiDEPT page. You will see a complete list of departments based on the DEPTAB file. (If the contents of the DEPTAB file are changed, you will need to rebuild the iiiDEPT component or you can manually edit the HTML.)
- 7. Use the Component menu category and select the Maintain option to review the Web components in the registry.
- 8. You should see that your iiiDEPT component is defined as INPUT mode. If you attempt to change this component, you can only modify the description and linked page.

9. Check that the iiiFN07 function has compiled successfully, and then execute your iiiFN07. You will see a drop down for selecting the department code field.

Using the drop down, select the Department Code of ADM and press the *Search* button.

When the Sections in the Department are displayed, the iiiDEPT field is not displayed using the iiiDEPT component (i.e. it is not displayed as a drop down) as the mode of the display is output. For more details, refer to Automatic Embedding of Web Components.

## Step 3. Create a Text Web Component

In this step, you will create a simple Text Web component which will indicate when the Web pages will be updated next. By using a Web component, you can centralize the definition so that only a single change needs to be made to your application.

- 1. Use the Components menu category and select the Maintain option. A list of all components in the partition will be displayed.
- 2. Press the *Add* button to define a new component as follows:

Name:	iiiUPDATE
Type:	Text

3. Press the *Continue* button and then enter the following details:

Description:	Next Update
Text:	 <center><strong>Pages will be updated on January 1, 2002.</strong></center>
Mode:	Not Applicable

This component is created with a Not Applicable mode because it will be used in the footer. It is not used in a REQUEST or DISPLAY.

- 4. Close the Component window.
- 5. Open the iiiPROC03\_STDFOOTER page. At the bottom of the document, add a LANSA tag to embed the component as follows:

<RDML COMPONENT="iiiUPDATE">

**Note:** The component names are case sensitive so be careful to use the correct spelling of the component .

Tip: You can use the Tags menu category and select the RDML -

Component... option if you wish to see a list of components.

- 6. Execute your iiiFN05 or iiiFN07 function to see the new information in the footer of the function.
- 7. Change the iiiUPDATE component so that the text reads as follows: <br /><center><strong>Pages will be updated on December 31, 2002. </strong></center>
- 8. Test your function again. The change is made immediately. No HTML editing or function recompiling was required.

#### **Step 4. Banner Web Component**

In this step, you will add an advertising banner to the standard header used in your functions. This banner will link to the Visa, MasterCard and LANSA Web Site. (**Note:** You may not be able to test the links if your Web Server does not have access to the Internet.)

1. Create a new component as follows:

Name:	iiiBANNER
Type:	BANNER

2. Press the *Continue* button and enter the following:

Type:	Image
Mode:	Not Applicable

3. Press the *OK* button and enter the following:

Description:	Function Banner
Update frequency:	Update every visit
Display link in new window:	Check the box

4. Using the *Add* button, enter the following data for the banner items:

Set	Image	Description	URL
0	LANSA.GIF	LANSA	http://www.lansa.com
1	VISA.GIF	Visa	http://www.visa.com
2	MCARD.GIF	MasterCard	http://www.mastercard.com

- 5. Press *OK* to save the component.
- 6. Open the iiiPROC03\_STDHEADER page.
- 7. Include the banner component at the bottom of the function header by inserting the following LANSA tag:

<RDML COMPONENT="iiiBANNER">

**Note:** The component names are case sensitive so be careful to use the correct spelling of the component .

8. Test your changes by executing your iiiFN05 or iiiFN07 function. You should see a new banner each time you perform a new search. (**Note:** You may not be able to test the links to other Web sites if your Web Server does not have access to the Internet.)

# Step 5. Use Hidden Fields in Function

In this step, you will add a hidden field to the REQUEST panel in iiiFN05 and assign a Web Page component to the field. This component will contain the HTML for displaying the CALENDAR button which you manually added to the HTML. Using this technique, the calendar button will automatically be included in the page without manually editing the HTML.

1. In the LANSA Editor, edit the RDML for your iiiFN05 function. Add the following RDML line to define a field in the function:

```
DEFINE FIELD(#iiiCOMP) TYPE(*CHAR) LENGTH(1)
```

The characteristics of this field are not important. The field is simply a dummy hidden field which can be used for component substitution.

2. Modify the REQUEST statement so that it appears as follows:

REQUEST FIELDS((#DEPTMENT)( #STD\_DATE) (#iiiCOMP \*HIDDEN)) DESIGN(\*DOWN) IDENTIFY(\*DESC) MENU\_KE

- 3. Save and exit the RDML function but do NOT compile the function yet.
- 4. Create a new HTML page by using the File menu category and selecting the *New* option.
- 5. Delete the default HTML which appears and enter the following HTML into the page:

```
<a href="javascript:CallCalendar()">
<img src="/IMAGES/TB_CAL.GIF" alt="Calendar" border="0" width="70" h
</a>
```

- 6. Save the file as iiiCOMP with description Hidden Field Component for Calendar.
- 7. Use the Components menu category and select the Maintain option. A list of all components in the partition will be displayed. Press the *Add* button to define a new component as follows:

Name:	iiiCOMP
Type:	PAGE

Press the *Continue* button and then enter the following details:

Description:	Calendar Button
Page:	iiiCOMP
Mode:	OUTPUT

The mode of this component is very important. Because the field is hidden in the REQUEST statement, this component must be an OUTPUT component.

- 8. Now that the component has been created, you can submit the iiiFN05 function for recompile. Remember, compiling the function will replace your modified HTML with a new version.
- 9. Check that the compile was successful and then test your function. You should see the Calendar button at the top of your function. Notice that you did not have to edit the HTML. The iiiCOMP was automatically embedded by LANSA. You may wish to review the HTML page for function iiiFN05 to review how the hidden field was used in the HTML page.

## Step 6. Create a Web Link Component

In this step, you will use a Web link component to control the flow between WEBEVENT functions. You will add an image to act as a link between iiiFN05 and iiiFN06. This step will also demonstrate how you can embed components within components.

1. Create a new component as follows:

Name:	iiiWEBLINK
Type:	Web Link

Press the *Continue* button and enter:

Description:	Link to iiiFN06
Linked Process:	iiiPROC03
Linked Function:	iiiFN06
Linked Description:	Next Function
Туре:	IMAGE
Image:	TB_NEXT.GIF

In order to add this component to your function, you will need to use an RDML COMPONENT tag. If you manually edit the HTML, your changes will be overwritten when you recompile the function. Instead, you will add the iiiWEBLINK component to an existing component.

2. Using the Web Function Editor, open the iiiCOMP page you created for the Calendar button.

Add the bolded line to your HTML so that it appears as follows:

```
<a href="javascript:CallCalendar()">
<img src="/IMAGES/TB_CAL.GIF" alt="Calendar" border="0" width="70" h
</a>
```

#### <RDML COMPONENT="iiiWEBLINK">

3. Save the document.

You have now embedded the iiiWEBLINK component into the iiiCOMP. When iiiCOMP is embedded into the HTML page, it will also embed the iiiWEBLINK component.

4. Test your iiiFN05 function. (You do not need to recompile the function.) You should see a *Next* button beside your *Calendar* button. Try using this button instead of the *Search* button.

#### Summary

#### **Important Observations**

- The contents or data used for a Visual Web component will be determined when the component is created. The Department Code drop down lists all departments in the DEPTAB file when the component was created.
- The text in components may contain HTML tags.
- Component can be embedded into components provided that there is no recursive embedding.
- Modes are very important to Web components especially when a component is embedded automatically by LANSA for the Web.
- In this tutorial, two fields were used for Web component substitution: iiiDEPT and iiiCOMP. iiiDEPT was created in the LANSA Repository but iiiCOMP was defined in the RDML function. It is strongly recommended that all field used in REQUEST or DISPLAY statements be created in the LANSA Repository so that a common field definition is used when data is automatically exchanged by LANSA for the Web. The iiiCOMP field was created to demonstrate the use of hidden fields for Web component substitution. These fields will never contain data. iiiCOMP is only used for embedding a Web component.

# **Tips & Techniques**

- Text Web components are very similar to text graphic variables. The main difference is that the graphic variables are defined at the system level. Web components are defined at the partition level.
- Remember, component names are case sensitive.
- Using a hidden field to automatically embed a component eliminates the need to manually add components to your HTML pages.
- Embedding Web components into other Web components is a very powerful development technique, but be very careful that you do not recursively embed components.
- For more examples of Web components, refer to the SET Collection.

# What I Should Know

- How to create Visual Web components.
- How to create Text, Banner, Web Link and Page Web components.

- How are modes used by Web components.
- How LANSA for the Web embeds components into HTML pages.
- How to embed Web components into other Web components.

# **Other Tutorials**

You are now ready to complete WEB009 - Web Page Substitution (Optional - Advanced). This tutorial is optional.

## WEB009 - Web Page Substitution (Optional - Advanced)

**Note:** If you copy and paste these HTML examples into the Web Function Editor, use HTML Paste rather than Paste, so that the HTML tags are copied with the text.

#### **Objective:**

- To demonstrate a technique which can be used so that custom changes to your HTML will not be replaced when a function is recompiled.
- To show how to use a single field and a Web page component to substitute for the display in a function so that changes to the HTML page are not lost when a function is recompiled and the HTML is regenerated.

This tutorial is optional. It introduces some more advanced level concepts, which are very important to developing WEBEVENT function applications with LANSA for the Web. This tutorial provides a very simple example to demonstrate the concept of Web page substitution.

To achieve these objectives, you will complete the following steps:

- Step 1. Create a New Function
- Step 2. Create a New Web Component
- Step 3. Copy the HTML Code For Page Component iiiFN08C
- Step 4. Test the Function
- Step 5. Add Fields to the iiiFN08 Function
- Optional Step 6. Modifying iiiFN06
- Summary

#### **Before You Begin**

You may wish to review these topics and all of their related sections:

- WEBEVENT Functions
- Web Components

In order to complete the tutorials, you must have completed the following:

- WEB002 Coding a WEBEVENT Functions
- WEB004 LANSA Generated HTML Pages

- WEB006 Graphic Variables
- WEB008 Web Components

## **Step 1. Create a New Function**

In this step you will create a new function which is a duplicate of your iiiFN05 function. The iiiFN08 function, like iiiFN05, will be used with the iiiFN06 function to display a listing of sections. (Remember to substitute iii with your initials.)

- 1. Working with your iiiPROC03 process, create a new function named iiiFN08 Request Department. (Tip: You may to copy your existing iiiFN05 function to start the iiiFN08 function.)
- 2. The RDML code in your iiiFN08 function should appear as follows:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
DEFINE FIELD(#iiiFN08C) TYPE(*CHAR) LENGTH(1)
CHANGE FIELD(#DEPTMENT) TO(*DEFAULT)
REQUEST
FIELDS((#iiiFN08C *NOID)) DESIGN(*DOWN) IDENTIFY(*DESC) MEN
REQUEST FIELDS((#DEPTMENT)
(#STD_DATE)) DESIGN(*DOWN) IDENTIFY(*DESC) MENU_KEY(*NO
Following are some important notes about this function:
```

The field iiiFN08C is simply created for Web component substitution. The field uses \*NOID as no description is required. In Step 2, you will create the

HTML for this page component.

This function has two REQUEST commands. Only the first REQUEST will be processed in the WEBEVENT function but HTML pages will be created for both commands. You will use the generated HTML from the second REQUEST to create the iiiFN08C Page component.

- 3. Do not compile your function until after you define your Web component.
- 4. Use the Tools menu category and select the Keywords Maintain option. Do not enter a Process. Press the *OK* button to continue. Press the *Add* button to create a link to by entering the following information:

Process	iiiPROC03
Function	iiiFN08
Keyword	SEARCH
Description	Search

Linked Process iiiPROC03

Linked Function **iiiFN06** 

#### Step 2. Create a New Web Component

You will now create a Web Page component named iiiFN08C which will be used for displaying the fields in the REQUEST panel. Once this component is created, the first REQUEST statement (which is used to create the display which is sent to the browser) will produce the following generated HTML:

```
<strong></strong><RDML COMPONENT="iiiFN08C " MODE="I"></RDML>
```

The iiiFN08C Web component will be created as a page Web component. The page will contain the HTML code for the fields in the second REQUEST statement, i.e. the screen with the fields you really want to send to the browser.

1. Create a new component as follows:

Name:	iiiFN08C
Type:	PAGE

Press the *Continue* button and enter the following:

Description:	HTML Page Layout for iiiFN8
Page:	iiiFN08C
Mode:	INPUT

Press the *OK* button to create the component.

2. Submit the function for compile.

When the function is compiled, it will include the Web component you have just defined.

The function compile will also create the HTML which will be used in the

iiiFN08C page component.

# Step 3. Copy the HTML Code For Page Component iiiFN08C

In this step, you will cut and paste the HTML code required for the iiiFN08C page component. The HTML is actually created for you by LANSA. The second REQUEST statement contains the HTML code required for the display.

- 1. Using the Web Function Editor, open a new document.
- 2. Delete the default HTML which appears in the new page.
- 3. Open the iiiPROC03 iiiFN080002 HTML page. (This is the second REQUEST statement.)

Copy all the HTML statements in the table definition, i.e. copy all code between the following tags

copy all HTML code here but do not include the table tags shown above and be

4. Paste the HTML code into your new page. Your code should appear something like this:

```
<t
```

- 5. Make the following changes to the HTML:
  - Add some text instructing the user to enter a Department Code a.
  - Manually type the names of the DEPTMENT and STD\_DATE fields b. instead of using RDML MERGE
  - c. Include the Calendar button using HTML
  - d. Manually type the names of thinclude the next button using your iiiWEBLINK component.

The final pages should appear as follows (changes are in bold):

```
<tr>
```

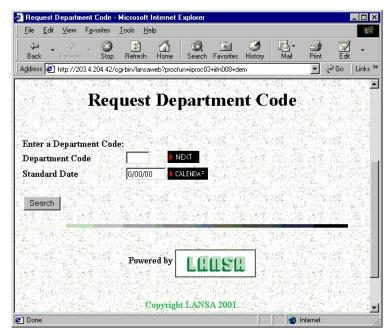
```
<strong>Enter a Department Code:</strong>
 Strong>Department Code<strong>
 <input name="ADEPTMENT " type="text" size="004" maxlength="004"
 value="<RDML MERGE="DEPTMENT ">"
 onfocus="SetNameLocation('ADEPTMENT ',03,29)" />
 Standard Date</strong>
 <input name="SSTD_DATE " type="text" maxlength="008"
 align="right" <RDML MERGE="STD_DATE " EDITCODEI="Y">
 onfocus="SetNameLocation('SSTD_DATE ',04,29)" />
 <a href="javascript:CallCalendar()">
 <img src="/IMAGES/TB_CAL.GIF" alt="Calendar" border="0" width=
 vspace="0" align="left" /></a>
```

6. Save the document as iiiFN08C with description iiiFN08 Page Component.

# Step 4. Test the Function

In this step, you will execute your iiiFN08 function. The iiiFN08 can be recompiled and the layout in your screen HTML will not be impacted

1. Execute your iiiFN08 function. The page should appear something like the following:



2. Edit the iiiFN08C page and add the bold code:

```
<input name="ASSTD_DATE " type="text" maxlength="008" align="right"
<RDML MERGE="STD_DATE " EDITCODEI="Y">
onfocus="SetNameLocation('SSTD_DATE ',04,29)" />
```

#### <strong>Press NEXT to search.<strong>

- 3. Save your changes.
- 4. Test your iiiFN08 function to see your changes. (Reminder: The iiiFN06 function is linked to the iiiFN05 function when you use the *New Search* button. You must use the browser Back button or change the keyword links to return to the iiiFN08 function.)
- 5. Edit the RDML code in your iiiFN08 function to remove the USER\_KEYS((01 SEARCH)). The Web Link component will be used instead. The REQUEST should now appear as follows:

```
REQUEST
FIELDS((#iiiFN08C *NOID)) DESIGN(*DOWN) IDENTIFY(*DESC) MEN
```

- 6. Recompile the iiiFN08 function.
- 7. Test your iiiFN08 function to see your changes.

The *Search* button should not appear.

Your text added in Step 2 is not impacted by the recompile because it is stored in the Web Page component!

# Step 5. Add Fields to the iiiFN08 Function

In this step, you will add another field to the REQUEST used in iiiFN08 to see how this will impact the iiiFN08C Web page component being used for the display.

1. Edit the RDML code in your iiiFN08 function. Add the STD\_NAME field to the CHANGE and REQUEST statements. Your RDML should appear as follows:

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
DEFINE FIELD(#iiiFN08C) TYPE(*CHAR) LENGTH(1)
CHANGE FIELD(#DEPTMENT #STD_NAME) TO(*DEFAULT)
REQUEST
FIELDS((#iiiFN08C *NOID)) DESIGN(*DOWN) IDENTIFY(*DESC) MEN
REQUEST FIELDS((#DEPTMENT)
(#STD_DATE) (#STD_NAME)) DESIGN(*DOWN) IDENTIFY(*DESC) MI
```

- 2. Compile the iiiFN08 function.
- 3. Open the iiiPROC03 iiiFN080002 HTML page. In this page, you will see the STD\_NAME field added to the list of input fields. You could copy this new code to your iiiFN08C component.
- 4. Edit the iiiFN08C page and add the bolded code (notice that the label for STD\_NAME has been replaced by the words Requested By):

```
<RDML MERGE="STD DATE "EDITCODEI="Y">
 onfocus="SetNameLocation('SSTD_DATE ',04,29)" />
 <a href="javascript:CallCalendar()">
 <img src="/IMAGES/TB_CAL.GIF"
 alt="Calendar" border="0" width="70" height="21" hspace="0" vspace="0" al
 </a>
 <strong>Requested By<strong>
 <input name="LSTD_NAME " type="text" size="025" maxlength="025'
 value="<RDML MERGE="STD NAME ">"
 onfocus="SetNameLocation('LSTD_NAME ',05,29)" />
```

- 5. Save your changes.
- 6. Test your iiiFN08 function.

# **Optional Step 6. Modifying iiiFN06**

In this step, you will modify the iiiFN06 function so that it displays the STD\_NAME field which was added to the REQUEST in iiiFN08.

1. Edit the RDML code in your iiiFN06 function to display the STD\_NAME field by changing the GROUP\_BY as follows:

GROUP\_BY NAME(#HEADER) FIELDS(#DEPTMENT #DEPTDESC #STD\_NAME)

- 2. Recompile your iiiFN06 function.
- 3. Execute your iiiFN08 function. Enter a Requested By name when you perform your search using the NEXT button. The name should appear in the list of sections displayed by iiiFN06.

#### Summary

#### **Important Observations**

- A WEBEVENT function may contain more than one REQUEST and/or DISPLAY statement. The HTML pages are created for each REQUEST and DISPLAY, but only the first statement will be processed. By definition, WEBEVENT functions terminate after a display is processed.
- Two REQUEST statements were used as a means of having LANSA generate the HTML for the fields. Once you are an experienced LANSA Web function developer, you may not require the second display statement. For documentation purposes, the second display statement is very helpful to other developers.
- In this example, a multilingual partition was used. The RDML MERGE tags (for example <RDML MERGE="&T0001+0001+0020">) for the field labels were removed and labels were manually entered. If you do not manually replace the field labels with text, you must update the MERGE tags after a function is recompiled.

## **Tips & Techniques**

- You can use this technique with almost any display. The screen can include input field, output fields, browse lists, etc.
- When using this technique, you should always use a REQUEST statement for the Web Page component. Your second statement (i.e. the actual screen layout) can be a REQUEST or DISPLAY. Using a REQUEST will simplify the HTML generated as modes are not used.
- You can link WEBEVENT functions using keywords or by using Web Link components. You may also use JavaScript and the HandleEvent function to control navigation in WEBEVENT functions.

# What I Should Know

• How to use a Web page component to replace the display so that recompiling the function does not impact the customized HTML.

# WEB010 - Web Skeletons (iSeries)

# **Objective:**

- In this exercise, you will modify LANSA's Web Skeletons in order to make a change to the default layout of the Web functions used in your Personnel System Web application. This exercise provides a simple example of how the Web Skeleton can be used to control the structure of the HTML generated by LANSA.
- To introduce LANSA's Web Skeletons. It is important to remember that there are many different uses for Web Skeletons. Web Skeletons simplify the design process. They do not eliminate it.
- To create a process level Web Skeleton. You will copy and then modify the default Web Skeleton in order to create a process level Web Skeleton for iiiPROC03.
- To recompile the functions in iiiPROC03 to use the new Web Skeleton.

To achieve these objectives, you must complete the following:

- Step 1. Copy Default Web Skeleton
- Step 2. Edit the Web Skeleton
- Step 3. Execute Your Function
- Step 4. Recompile Your Function
- Step 5. Execute Your Functions
- Summary

# **Step 1. Copy Default Web Skeleton**

In this step, you will copy the default Web Skeleton from DC@F28 and create a process-specific Web Skeleton in DC@W22.

1. Locate the Default Web Skeleton.

From an OS400 command entry, type in STRPDM and press Enter.

Select option to Work with members.

From the Work with Members screen, enter the following:

File	DC@F28
Library	<lansa data="" library=""> (default is DC@DTALIB)</lansa>
Member Name	WEBSKEL
Туре	TXT

2. Copy the Web Skeleton to DC@W22.

Beside member WEBSKEL, select option 3 to copy and press Enter. On the Copy Members screen, enter the following:

To File	DC@W22
New Name	iiiPROC03

- 3. Locate the copied Web Skeleton in DC@W22.
- 4. At the top of the Work with Members Screen, change the File to read DC@W22 instead of DC@F28 and press Enter.
- 5. Press F17 to Change using SDA
- 6. Change Member to iiiPROC03 and press Enter.

#### Step 2. Edit the Web Skeleton

In this step, you will simply change the copied Web skeleton, by removing the standard footer.

- 1. Edit your iiiPROC03 Web skeleton (using option 2) and press Enter.
- 2. Review the Web skeleton. It should appear as follows:

```
<RDML LAYOUT>
<!-- Process : %PR
                   %PRDS
<!-- Function : %FU
                   %FUDS
<!-- Page
        : %PG
<!-- Generated by - %PROD
<!-- Created by user - %USERI
<!-- Time and Date - %STAMP
<!-- RDML function sequence number - %SEQ
<!-- This is a *WEBEVENT function %IFWEV
                 %IFWEV
%W3FRM
<RDML MERGE="&HIDDEN">
<RDML COMPONENT="STDHEADER">
%W3TTL
\langle BR \rangle
<RDML MERGE="&MESSAGES">
<FONT SIZE="BASEFONT">
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3>
                                                        %IFN
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3 ALIGN=RIGH
<TBODY>
%W3FLD
</TBODY>
</TABLE>
<BR CLEAR="right">%IFRLTB
</FONT>
\langle BR \rangle
%W3BRW
<RDML MERGE="&BUTTONS">
<RDML COMPONENT="STDFOOTER">
</FORM>
```

- 3. Remove the Standard Footer by deleting the line <RDML COMPONENT="STDFOOTER">.
- 4. Insert the following, where the STDFOOTER used to be:

```
<FONT COLOR="RED"><CENTER><H3>Personnel System</H3></CENTER></FONT>
```

The bottom half of the Web Skeleton should now appear as follows:

```
...
%W3FLD
</TBODY>
</TABLE>
<BR CLEAR="right"> %IFRLTB
</FONT>
<BR>
%W3BRW
<RDML MERGE="&BUTTONS">
```

```
<FONT COLOR="RED"><CENTER><H3>Personnel System</H3></CENTER></FONT>
```

</FORM>

5. Press the F3 Exit key, to the exit the screen. The Change/Create member option should be defaulted to Y. Press Enter to save the changes.

#### **Step 3. Execute Your Function**

You will now execute the function before recompiling the function to use the new Web Skeleton. Your existing function's HTML was created using the default Web Skeleton.

1. Execute function iiiPROC03/iiiFN07 from your browser using the following URL:

http://<host address>/cgi-bin/lansaweb? procfun+iiiPROC03+iiiFN07+partition+language

2. Notice that the Standard footer is used, because the function has not been recompiled.

## **Step 4. Recompile Your Function**

You will now recompile the function to use the new process-specific Web Skeleton.

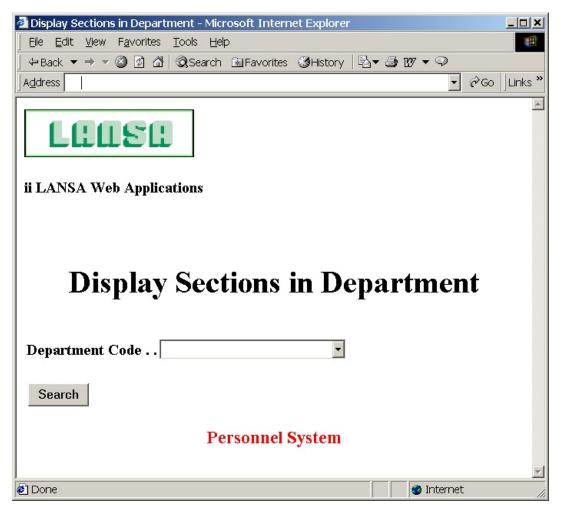
- 1. Recompile the function iiiPROC03/iiiFN07.
- 2. Also, recompile the function iiiPROC00/iiiFN00. This function should not be impacted as the Web Skeleton is process-specific.
- 3. Ensure that the compiles completed successfully.

#### **Step 5. Execute Your Functions**

1. Execute your iiiFN07 function, or if the browser is still open to the function, press the Refresh/Reload button in the browser.

Notice that the footer has been replaced, as the new Web skeleton is used.

Your function might appear something like the following:



2. Execute your iiiFN00 function. Notice that this function is not changed as the Web Skeleton is process-specific.

#### Summary

#### **Important Observations**

- The Web Skeleton can be used to modify the appearance of all function pages under a specific process.
- A function must be recompiled so that the HTML is regenerated in order for the Web Skeleton changes to take effect.
- The new Web Skeleton simply omitted the STD\_FOOTER from all Web functions in iiiPROC03. This change cannot be made using the Wizard. It cannot be done manually, but it would require editing the HTML for each function. When a change is required to all functions, the Web Skeleton is a good solution.

## **Tips & Techniques**

- The Web Skeleton is usually modified at the partition level to give all pages in the partition the same look and feel.
- NEVER modify the default Web Skeleton. When a new version of LANSA for the Web is installed, the default Web Skeleton could be changed.

## What I Should Know

- What the Web Skeleton is.
- How the Web Skeleton is used by LANSA when generating the HTML for a function.
- How to create and edit a process level Web Skeleton.
- How to change the appearance of your Web pages using the Web Skeleton.

## WEB011 - Using DEFAULT\_HIDDEN

#### **Objective:**

- To create a simple Web site login screen. The login will simply be used to identify the user so that you can customize or personalize the Web site to their ID. (This login example has not been created to emphasize Web site security.)
- To store the value of the login field using a process level \_HIDDEN so that it can be shared or passed from function to function.
- To demonstrate how to use <RDML PAGE=""> tag to simplify and modularize your page structure.

To achieve these objectives, you must complete the following:

- Step 1. Create Login Function
- Step 2. Edit Standard Header for iiiPROC03
- Step 3. Test your Functions
- Step 4. Create a Process Level \_HIDDEN Page
- Step 5. Test your Functions
- Summary

## **Step 1. Create Login Function**

In this step, you will create a new process iiiPROC04 which contains a login function iiiFN10. The login function simply requests a user ID. (You will create a new field in the Repository for the login ID.) The login ID will be added to the STDHEADER so that it is displayed in each function. The login function will have a link to the iiiFN07 Display Sections function. The iiiFN07 function will be modified to check that a login has been entered. If no value has been entered, the iiiFN10 login will be called so that the user must be logged in to use iiiFN07.

1. From the LANSA development environment, work with the fields in the LANSA Repository. Create a new field as follows:

Field Name	iiiLOGIN (where iii is your course assigned ID)
Туре	А
Length	10
Description	Login:

- 2. Create a new LANSA process named iiiPROC04 Login Process, where iii is your course assigned ID. (If the process already exists, select a different set of characters for iii.)
- 3. Enable your iiiPROC04 process for web. If you need to know how, refer to Web Enabling a LANSA Process.
- 4. Working with your iiiPROC04 process, create a new function named iiiFN10 Request Login. You will manually enter the code for the function.
- 5. Write the RDML Code to request the field iiiLOGIN. (This example does not include any field/file level validation. You could include validation routines inside the RDML to validate the login.)

One possible solution to this exercise is shown below (where iii is your course assigned ID):

```
FUNCTION (*DIRECT *WEBEVENT)
REQUEST
```

FIELDS((#iiiLOGIN)) EXIT\_KEY(\*NO) MENU\_KEY(\*NO) USER\_KEYS(

- 6. Exit and save your RDML function.
- 7. Compile your functions.
- 8. Using the LANSA HTML editor, register the keywords to link your iiiFN10 Request Login function to your iiiFN007 Display Sections in Department function

The keyword entries are specified as follows (where iii is your course assigned ID):

Process	iiiPROC04
Function	iiiFN10
Keyword	SUBMIT
Description	Login
Linked Process	iiiPROC03
Linked Function	iiiFN07

9. Working with your iiiPROC03 process, manually edit your function iiiFN07 so that it checks the value of iiiLOGIN. If the value is blank, then the iiiFN10 login function should be called. Your modified function might appear as follows (where iii is your course assigned ID):

```
FUNCTION OPTIONS(*DIRECT *WEBEVENT)
GROUP_BY
NAME(#HEADER) FIELDS((#iiiDEPT) (#DEPTDESC) (#RENTRY *HIDD]
DEF_LIST
NAME(#iiiLIST) FIELDS((#LISTDUMMY *HIDDEN) #SECTION #SECDE
IF COND('#iiiLOGIN *EQ *BLANKS')
CALL PROCESS(*DIRECT) FUNCTION(iiiFN10)
ENDIF
```

```
IF COND('#RENTRY *NE Y')
```

•••

- 10. Exit and save your RDML function.
- 11. Compile your function.

## Step 2. Edit Standard Header for iiiPROC03

In this step, you will edit the standard header page to be used specifically with iiiPRO0C03 to include the iiiLOGIN field.

- 1. Using the LANSA HTML editor, open the iiiPROC03\_STDHEADER.
- 2. Insert the bolded lines which will simply add a heading to the page (where iii is your course assigned ID):

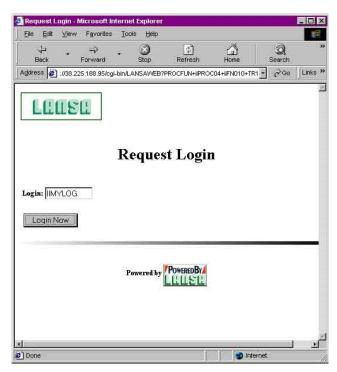
```
<!--
                            -->
<!--
          LANSA for the WEB
                                     -->
<!--
          Standard Header
                                  -->
<!--
                            -->
<IMG SRC="
<RDML MERGE="*LW3CPYLOGO">" WIDTH=221 HEIGHT=62>
<P>
<H4> iii LANSA Web Applications</H4>
<P>
Your login entered was: <RDML MERGE="iiiLOGIN">
<P>
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=100%
<TBODY>
•••
```

4. Save the document.

## **Step 3. Test your Functions**

- 1. Check that your functions compiled successfully.
- 2. Execute your function iiiPROC04/iiiFN10.

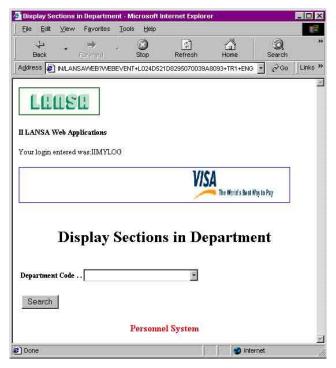
Enter iiiMYLOG as your login ID (where iii is your course assigned ID).



3. Press the Login Now button.

Remember: Your login ID is automatically transferred to iiiFN07 because it is an input field.

4. The iiiFN07 Display Sections function will be displayed.



5. Select a department and press the Search button.

Notice that you are returned to function iiiFN10 Request Login because the iiiLOGIN field contained in the standard process header is an output only field. The value is not passed back to function iiiFN07 when the function attempts to display the sections in the selected department.

#### Step 4. Create a Process Level \_HIDDEN Page

In this step, you will create a new HTML page called iiiPROC03\_HIDDEN. You will include iiiLOGIN field in this page so that the value will then be passed to all functions contained in your iiiPROC03 process. You will use the <RDML PAGE=""> command to include standard DEFAULT\_HIDDEN as part of this page.

- 1. Using the LANSA for the Web editor, create a new HTML page.
- 2. When the new page appears, delete all of the default HTML code.
- 3. Add the following lines to the page:

<RDML PAGE="DEFAULT\_HIDDEN"> <INPUT NAME="AiiiLOGIN" TYPE="HIDDEN" SIZE="10" VALUE=" <RDML MERGE="iiiLOGIN">">

Note that 'A' must precede AiiiLOGIN, to indicate that the field is alphanumeric.

4. Save your document as iiiPROC03\_HIDDEN.

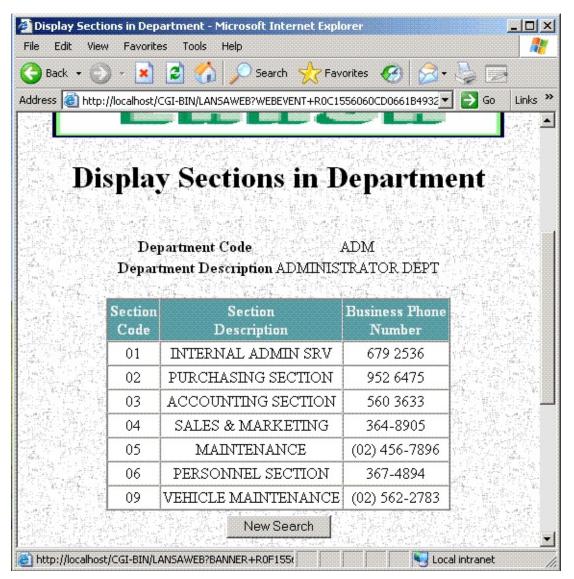
#### **Step 5. Test your Functions**

In this step, you will retest your functions after creating the iiiPROC03\_HIDDEN page. Notice that you do not need to recompile your functions or edit any of the function HTML.

1. Execute your function iiiPROC04/iiiFN10.

Enter iiiMYLOG as your login ID (where iii is your course assigned ID) and press the Login button.

2. Select a department and press the Search button.



Notice that the value of iiiLOGIN is now returned to the iiiFN07 function.

- View the HTML source for the iiiFN07 function.
   Notice that iiiLOGIN field has been passed as a hidden field.
- 4. Re-execute your iiiFN10 Request login function.Enter a different value for the login and press Login.Notice that the new value is passed.

#### Summary

#### **Important Observations**

- Only input capable fields, hidden fields or browse lists are passed from function to function. The iiiLOGIN field was passed from iiiPROC04/iiiFN10 to iiiPROC03/iiiFN07 as an input capable field.
- The iiiLOGIN field was not passed from iiiFN07 back to iiiFN07 (reentrant function) because it was not an input capable or hidden field. By creating iiiPROC03\_HIDDEN, the field could be exchanged to any functions in iiiPROC03.
- Notice that the iiiFN07 function does not include the iiiLOGIN field in the REQUEST or DISPLAY statements, but the field can still be used by the function and in the HTML STDHEADER.
- The iiiLOGIN field must be used somewhere in iiiFN07 (for example, the IF statement or in a GROUP\_BY) for the value of iiiLOGIN to be stored in the function. Once it is stored in the function, it can be exchanged with the next function using the iiiPROC03\_HIDDEN fields.
- The Login function iiiFN10 is part of iiiPROC04. If you were to link iiiFN07 back to iiiFN10, the iiiLOGIN would not be passed because iiiPROC03\_HIDDEN is process-specific. iiiPROC04 still uses DEFAULT\_HIDDEN. Also, iiiFN10 has iiiLOGIN defined as an input field so it cannot use an exchanged value in \_HIDDEN. The iiiLOGIN field would have to be passed as a function parameter, browse list value or an input capable field on a display.
- Using the RDML PAGE tag allows you to include the standard DEFAULT\_HIDDEN page in your iiiPROC03 \_HIDDEN page. You could have copied the DEFAULT\_HIDDEN page to iiiPROC03\_HIDDEN and added the iiiLOGIN field, however any future changes made to DEFAULT\_HIDDEN would also have to be made to iiiPROC03\_HIDDEN. Using the RDML PAGE tag simplifies the maintenance of your code.
- Changes to the DEFAULT\_<pages> do not require the functions to be recompiled. These pages are automatically embedded when the function executes.

## **Tips & Techniques**

• Instead of adding iiiLOGIN to the iiiPROC03\_HIDDEN, it could have been added as a \*HIDDEN field in each REQUEST or DISPLAY.

- Creating process-specific pages allows you to customize your functions in a specific process. You should not modify the DEFAULT pages unless the change applies to all functions in the partition.
- All process-specific pages will automatically be exported with the process definition. These pages do not need to be registered as Web Page components.
- The DEFAULT\_HIDDEN page can be used to pass many values. The hidden field must be used in the RDML function in order for the value to be stored and passed to the next function. The field does not have to appear in a REQUEST or DISPLAY statement.
- The <RDML PAGE=""> tag can be used to modularize your code and simplify maintenance.

## What I Should Know

- How to use DEFAULT\_HIDDEN to store values to be passed from function to function within a process.
- How to use the <RDML PAGE=""> tag to create modular code.

### WEB012 - Dynamic Components

#### **Objective:**

- In this exercise, you will use the login field to add a dynamic component to your function. The dynamic component is determined by the value of a field instead of being coded into the HTML page. In this example, the data entered in the login field will be used to determine the component. This technique allows you to personalize the site based on the user login.
- To demonstrate how to include a page component using &FLD\_<field name>, which uses the value of the field as your component name instead of the field name.
- To demonstrate how to use components so that HTML changes are not lost when a function is re-compiled.

To achieve these objectives, you must complete the following:

- Step 1. Create a New Page Component
- Step 2. Modify iiiPROC03\_STDHEADER
- Step 3. Test Your Function
- Summary

#### Step 1. Create a New Page Component

In this step, you will create a new page component that will be used as the welcome. A very simple HTML page containing your name will be created. (Using a Web publishing tool, this HTML page could be made very graphical.)

1. Using the LANSA for the Web editor, create a new HTML page.

2. When the new page appears, delete all of the default HTML code.

3. Add the following line to the page:

```
<CENTER><STRONG>Hello (your name goes here)</STRONG>
</CENTER>
```

- 4. Save your document as iiiMYLOG with description Login Component. (Remember: iiiMYLOG is the data entered by the user when they login.)
- 5. Using the LANSA for the Web editor, open the maintain components window.
- 6. Select Add, to add a new component as follows (where iii is your course assigned ID):

Component	iiiMYLOG
Туре	PAGE

Description	Login Welcome
Page	iiiMYLOG
Mode	Not Applicable

7. Close the Maintain Components window.

## Step 2. Modify iiiPROC03\_STDHEADER

In this step, you will modify the HTML for iiiPROC03\_STDHEADER to use a dynamic component based on the contents of the iiiLOGIN field. When iiiLOGIN has a value of iiiMYLOG, it will include the component you created in Step 1. Create a New Page Component.

- 1. Using the LANSA for the Web editor, open the iiiPROC03\_STDHEADER (where iii is your course assigned ID).
- 2. Insert the bolded lines which will simply add a heading to the page:

```
<!--
                              -->
 <!--
            LANSA for the WEB
                                      -->
 <!--
            Standard Header
                                   -->
 <!--
                              -->
 <IMG SRC="
 <RDML MERGE="*LW3CPYLOGO">" WIDTH=221 HEIGHT=62>
 <P>
 <H4> iii LANSA Web Applications</H4>
 <P>
 Your login entered was: <RDML MERGE="iiiLOGIN">
<P>
 <RDML COMPONENT="&FLD_iiiLOGIN">
 <P>
 <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=100%
 <TBODY>
 •••
```

3. Save the document.

#### **Step 3. Test Your Function**

1. Execute your function iiiPROC04/iiiFN10.

Enter iiiMYLOG as your login ID (where iii is your course assigned ID) and press the Login button.

2. Notice how the value of the iiiLOGIN field is used to display the new component iiiMYLOG.

🛎 Disp	lay Se	ections	in Departr	nent -	Microso	ft Internet Explo	orer		_ 🗆 ×
<u>F</u> ile	<u>E</u> dit	⊻iew	F <u>a</u> vorites	<u>T</u> ools	<u>H</u> elp				1
	ф ack	•	➡ Forward	*	Stop	🖈 Refresh	Home	Q Search	**
Addres	s 🦉	/LANS/	WEB?WEBI	EVENT	+L027578	31A49D58300A4D	8094+TR1+ENG	▼ ∂Go	Links »
L	.61	15	6						Á
$\Pi L$	ANSA	A Web	Applicat	ions					
Your	login	entere	ed was:⊞v	IYLO	G				
			]	Hello	(your 1	name goes he	re)		
						E	Ŀ		
	Di	isp	lay S	ect	tion	s in D	epartn	nent	
Dep	artm	ent Co	ode				•		_
Se	arch	J							•
🎒 Don	e						🔮 Inte	rnet	

3. Using iiiFN10, try entering any other registered component in the login field. For example, try your iiiBANNER component.

#### Summary

#### **Important Observations**

- &FLD\_<field name> components allow you to determine the component name based on the value of a field. In this example, the field value was based on user input. The value could also be set based on information in a database file.
- The importance of &FLD\_<field name> is that you can use RDML to control the value of the field during execution of the function.

## **Tips & Techniques**

- In some cases, you can use &FLD\_<field name> as an alternative to <RDML CONDITION> or <RDML NOTCONDITION> tags. Instead of using conditional tags, you can control HTML components by setting the field values (in the RDML) used in the &FLD\_<field name>. For example, IF a certain condition is met, then set the field value to COMP1, where COMP1 is a component with the desired HTML.
- You can embed components within components, provided that they are not called recursively.
- In this example, you created a very simple personalization based on the user login. You could create a highly personalized Welcome screen if you maintain additional data in a database for the login user ID. For example, you could display the last time a user visited the site or you could show a list of user preferences, etc.
- Create page components to protect your HTML changes from being lost when a function is recompiled.
- Using &FLD\_<field name> allows you to customize what a user/users might see for the same page, depending on their login authority.
- Remember to use the proper mode when defining your Web components. The iiiMYLOG component uses mode Not Applicable because it appears in the function header.

## What I Should Know

- How to use &FLD\_<field name> to dynamically set a Web component based on the value of a field.
- How to embed a component within a component.

• How to create page components to protect HTML from being replaced when a function is recompiled.

## WEB013 - JavaScript and Browse Lists

To achieve the results shown in this tutorial, you will need ten graphic files as described in step 5 of the iSeries Installation Steps in Tutorials for Web Functions & WEBEVENTs.

If you do not have these files or if they are not specified in the PSLMST file, you will simply see a box with an X, instead of a picture.

## **Objective:**

- In this exercise, you will modify the Display Sections inquiry function to allow the user to display a list of all employees in a selected Department and Section. You will allow the user to select a specific Department and Section (a row in the browse list) to call a Employee List function. The Employee List function will include an employee picture (in place of the PHONEBUS field). This image will also be turned into a link, that will open a new browser window containing the details for the selected employee.
- To create a new JavaScript function which will pass parameters from the browse list to the HandleEvent function.
- To demonstrate the use of JavaScript to pass function parameters.
- To demonstrate the ability to select a row in a browse list.
- To demonstrate how to call a LANSA function and pass parameters or funcparms.
- To create an Employee Details function using the Web page substitution techniques used in exercise WEB09.

To achieve these objectives, you must complete the following:

- Step 1. Create Fields to be used as Function Parameters
- Step 2. Create iiiFN11 Employee List
- Step 3. Edit Process Level \_SCRIPT
- Step 4. Create a Link Component
- Step 5. Modify Function iiiFN07
- Step 6. Test Changes
- Step 7. Include New Component Link into Employee Browse List
- Step 8. Build Function iiiFN12 Employee Details
- Step 9. Test Your New Functions
- Step 10. Execute function iiiFN12 from a Command Line (Optional)

- Step 11. Create Page Component for Function iiiFN12 (Optional)
- Step 12. Test Changes (Optional)
- Summary

#### **Step 1. Create Fields to be used as Function Parameters**

In this step, you will create two fields in the LANSA Repository. The iiiDEPTWK and iiiSECTWK will be used to store the selected Department and Section. These fields will also be added to the <process\_hidden> HTML you created in an earlier exercise. These fields will be required by the JavaScript written later in the exercise.

1. From the LANSA Development Environment, work with the fields in the LANSA Repository. Create two new fields as follows (where iii is your course assigned ID):

Field Name	iiiSECTWK	
Reference Field	SECTION	

Field Name	iiiDEPTWK	
Reference Field	DEPTMENT	

- 2. Using your LANSA for the Web HTML editor, open iiiPROC03\_HIDDEN.
- 3. Add the following lines (where iii is your course assigned ID):

<INPUT NAME="AiiiSECTWK" TYPE="HIDDEN" SIZE="2" VALUE=' <RDML MERGE="iiiSECTWK">"> <INPUT NAME="AiiiDEPTWK" TYPE="HIDDEN" SIZE="4" VALUE=' <RDML MERGE="iiiDEPTWK">">

4. Save your document.

## Step 2. Create iiiFN11 Employee List

In this step, you will create a function to display information (EMPNO, GIVENAME, SURNAME) about the employees in a given department and section. The values for department and section are passed in the iiiDEPTWK and iiiSECTWK fields in the iiiPROC03\_HIDDEN.

- 1. Working with your iiiPROC03 process, create a new function named iiiFN11 Employee List. You will manually enter the code for the function.
- 2. Write the RDML Code to build a browse list from the two parameters passed, iiiDEPTWK and iiiSECTWK. (This example does not include any field/file level validation. You could include validation routines inside the RDML to validate that records were found for the browse list.)

#### Solution:

One possible solution to this exercise is shown below (where iii is your course assigned ID):

```
FUNCTION (*DIRECT *WEBEVENT)
DEF_LIST
NAME(#iiiEMPLST) FIELDS((#EMPNO) (#GIVENAME) (#SURNAME))
IF COND('#iiiLOGIN *EQ *BLANKS')
CALL PROCESS(*DIRECT) FUNCTION(iiiFN10)
ENDIF
CLR_LIST NAMED(#iiiEMPLST)
SELECT
FIELDS((#iiiEMPLST)) FROM_FILE(PSLMST1) WITH_KEY(#iiiDEPTWK
ADD_ENTRY TO_LIST(#iiiEMPLST)
ENDSELECT
DISPLAY
FIELDS((#iiiDEPTWK) (#iiiSECTWK)) BROWSELIST(#iiiEMPLST) EXIT_
```

- 3. Exit and save your RDML function.
- 4. Compile your RDML function iiiPROC03/iiiFN11.

## Step 3. Edit Process Level \_SCRIPT

In this step, you will edit your iiiPROC03\_SCRIPT. You will add a new JavaScript function which will be used to store the Department and Section values (using iiiPROC03\_HIDDEN) before the next function is called using the HandleEvent script shipped with LANSA.

- 1. Using the LANSA for the Web editor, open iiiPROC03\_SCRIPT.
- 2. Add the following lines to an appropriate part of the page (where iii is your course assigned ID):

```
function HandleEventDeptSect(Proc,Func,Dept,Sect)
{
    document.LANSA.AiiiDEPTWK.value=Dept;
    document.LANSA.AiiiSECTWK.value=Sect;
    HandleEvent(Proc,Func);
}
```

4. Save your document.

### Step 4. Create a Link Component

In this step, you will create a page component that will appear as a button to link to another function. The button will use the HandleEventDeptSect JavaScript function created in Step 3 to set the value of the work fields and then call the next function.

- 1. Using the LANSA for the Web editor, create a new HTML page.
- 2. When the new page appears, delete all of the default HTML code.
- 3. Add the following lines to the page (where iii is your course assigned ID):

```
<INPUT TYPE="button" VALUE="Click Here" NAME="button1"
onClick="HandleEventDeptSect('iiiPROC03','iiiFN11',
'<RDML MERGE="DEPTMENT">','<RDML MERGE="SECTION">');">
```

NOTE: there is no space following any of the commas. Remember to use uppercase characters for your iii identifier.

- 4. Save your document as iiiLNK with description Link Component.
- 5. Using the LANSA for the Web Editor, open the Maintain Components window.
- 6. Press Add, to add a new component as follows (where iii is your course assigned ID):

Component	iiiLNK
Туре	Page

Description	Link Component
Page	iiiLNK
Mode	OUTPUT

## Step 5. Modify Function iiiFN07

In this step, you will make changes to function iiiFN07. First, you will change the browse list name from iiiLIST to iiiLIST1 so that you can customize the appearance of this specific browse list using the Web Functions Wizard . Next, you will define an iiiLNK field which will be added to the list of Sections browse list. This field will be used for the link component iiiLNK to call the Employee list function. Finally, you will add the DEPTMENT and iiiLNK fields to the browse list.

- 1. Working with your iiiPROC03 process, edit the function named iiiFN07 Display Sections in Department.
- 2. Using the find and change facilities in the RDML editor, change all occurrences of iiiLIST to iiiLIST1.
- 3. Add a DEFINE statement for the iiiLNK field as follows:

#### DEFINE

FIELD(#iiiLNK) TYPE(\*CHAR) LENGTH(1) COLHDG(Find Employee)

4. Add the field DEPTMENT with an attribute of \*HIDDEN to your browse list (now called iiiLIST1). Also add your new field iiiLNK with an attribute of \*NOID to your browse list. Also, change the number of entries allowed for the list to 9999. The RDML code should appear as follows:

DEF\_LIST

NAME(#iiiLIST1) FIELDS((#LISTDUMMY \*HIDDEN) (#SECTION) (#SEC

- 5. Exit and save your RDML function.
- 6. Compile your function.

## **Step 6. Test Changes**

In this step, you will test your function and HTML documents that you created in the previous steps.

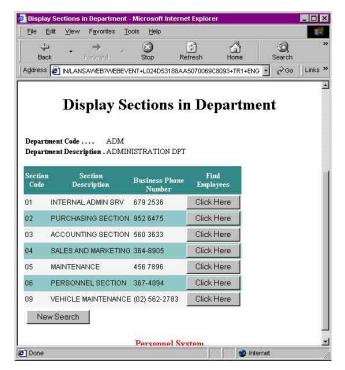
1. Execute your function iiiPROC04/iiiFN10.

Login using iiiMYLOG.

Select the Administration department and press the SEARCH button.

2. Notice the new button on the right column of your browse list. It has the heading Find Employee, and says *Click Here*.

If these buttons are not properly aligned, check the code in the button component found in Step 3.



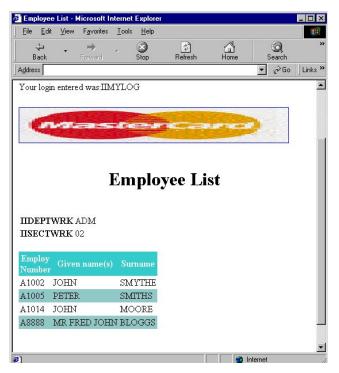
3. Clicking on any of the buttons in the browse list will call the iiiFN11 Employee List function.

**Tip:** If the link does not work, check the bottom left corner of your browser (Internet Explorer) for a tiny triangle with an apostrophe in it. This symbol indicates that there is an error in your JavaScript. Double click this icon, to view the error message.

**Note:** Most JavaScript errors occur due to typing errors. Check this first by comparing the Java Script code to the code in Step 3. Edit Process Level

\_SCRIPT.

4. When function iiiFN11 is called, a list of all employees in the selected Department and Section will be displayed:



If there are no employees, repeat the above procedure selecting a different Department and Section.

#### **Step 7. Include New Component Link into Employee Browse List**

In this step, you will create a component that will contain a link using an <A HREF> and an image (based on the value of the PHONEBUS field). This component will be included in the browse list used in function iiiFN11 and it will be used to call an Employee Details function (iiiFN12).

- 1. Using the LANSA for the Web editor, create a new HTML page.
- 2. When the new page appears, delete all of the default HTML code.
- 3. Add the following lines to the page (where iii is your course assigned ID and ppp=partition):

<a href="/cgi-bin/lansaweb?procfun+iiiproc03+iiifn012+ppp+funcparms+ EMPNO(A0050):<RDML MERGE="EMPNO">" target="New Employee"> <img src="/IMAGES/<RDML MERGE="PHONEBUS">" border="0"> </a>

Reminder: The <A HREF> link will not automatically exchange the WEBEEVENT data as it uses a new Web job. The Employee Number parameter is passed as part of the function call so that the iiiFN12 Employee Details function can retrieve the required data from the PSLMST file. The iiiFN12 Employee Details function is also opened in a new browser window.

- 4. Save your document as iiiIMAGE with description Employee Image.
- 5. Using the LANSA for the Web editor, open the Maintain Components window.
- 6. Select Add, to add a new component as follows (where iii is your course assigned ID):

Component	iiiIMAGE
Туре	Visual

Description	Employee Image Component		
Page	iiiIMAGE		
Mode	OUTPUT		

- 7. Working with your iiiPROC03 process, edit the function named iiiFN11 Employee List.
- 8. Define a new field iiiIMAGE based on the #PHONEBUS field and add the iiiIMAGE field to the browse list iiiEMPLST. Also add the PHONEBUS (which contains the name of the employee image graphic file name) as a \*HIDDEN field.

Your function should appear as follows (where iii is your course assigned ID):

FUNCTION (\*DIRECT \*WEBEVENT) DEFINE FIELD(#iiiIMAGE) REFFLD(#PHONEBUS) DEF\_LIST NAME(#iiiEMPLST) FIELDS((#iiiIMAGE \*NOID) (#EMPNO) (#GIVENAM CLR\_LIST NAMED(#iiiEMPLST) SELECT FIELDS((#iiiEMPLST)) FROM\_FILE(PSLMST1) WITH\_KEY(#iiiDEPTWK ADD\_ENTRY TO\_LIST(#iiiEMPLST) ENDSELECT DISPLAY FIELDS((#iiiDEPTWK) (#iiiSECTWK)) BROWSELIST(#iiiEMPLST) EXIT\_

- 9. Exit and save your RDML function.
- 10. Compile your function.

## Step 8. Build Function iiiFN12 Employee Details

In this step, you will build the Employee Details function, iiiFN12. This function will receive the employee number (EMPNO) from the calling function by using the funcparms passed in the URL. It will then display the Employee details.

- 1. Working with your iiiPROC03 process, create a new function named iiiFN12 Employee Details. You will manually enter the code for the function.
- 2. Write the RDML code to fetch the employee's data from the PSLMST file based on the value of EMPNO, which is a parameter passed to the function.

```
Include a GROUP_BY statement so that the iiiLOGIN field is defined in the function.
```

(This example does not include any field/file level validation. You could include validation routines inside the RDML to validate that records were found for the browse list.)

## Solution:

One possible solution to this exercise is shown below (where iii is your course assigned ID):

```
FUNCTION (*DIRECT *WEBEVENT)
******* This Group by can be left out however your login
******** component will not appear without it.
GROUP_BY NAME(#LOGGROUP) FIELDS((#iiiLOGIN))
GROUP_BY
NAME(#EMPLOYEE) FIELDS((#EMPNO *OUTPUT) (#GIVENAME *OU'
FETCH
FIELDS((#EMPLOYEE)) FROM_FILE(PSLMST) WITH_KEY(#EMPNO) IC
REQUEST FIELDS(#EMPLOYEE) EXIT_KEY(*NO) MENU_KEY(*NO)
Notice that this RDML solution uses a REQUEST statement instead of a
DISPLAY statement. This technique is used to reduce the amount of HTML
generated by LANSA for the Web.
```

- 3. Exit and save your RDML function.
- 4. Compile your RDML function iiiPROC03/iiiFN12.

#### **Step 9. Test Your New Functions**

In this step, you will test your new functions. You will execute function iiiFN07 as before. When the Employee List function iiiFN11 executes, you should see an employee image. When you click on an image, a new browser window should appear containing the Employee Details function iiiFN12.

- 1. Execute your function iiiPROC04/iiiFN10.
- 2. Login using IIIMYLOG.
- 3. Select the Administration department and press the SEARCH button.

Notice that the *Click Here* button appears exactly as it did in the previous exercise.

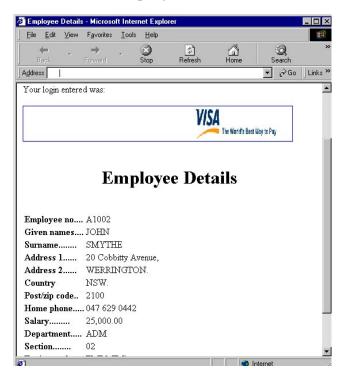
- 4. Select Section 02.
- 5. Press the *Click Here* button for the Section.

An employee list should appear. If the images have been placed correctly on the Web Server, an employee image should appear to the left of each name, otherwise you should see a small box with an X in it. (This small box is the standard browser default when an image cannot be found.)



5. Click on the employee image. (If no image exists, click on the box with the X.)

6. A new browser window should appear as function iiiFN12 executes. The window will contain the employee details:



7. Notice that the function header is not displayed as your iiiLOGIN field has no value. The hidden fields (including all of \_hidden) are not passed when using an <A HREF> link. You must include the fields as function parameters when using this type of link.

# Step 10. Execute function iiiFN12 from a Command Line (Optional)

In this step, you will call function iiiFN12 directly using a URL from the browser. This step is similar to using the <A HREF> link. It will allow you to call a function directly and practice parameter passing. You will add the iiiLOGIN parameter to the funcparms being passed.

1. Open a new browser window. To execute iiiFN12 directly, enter the following URL:

http://<server address>/cgi-bin/lansaweb? procfun+iiiproc03+iiifn012+ppp+funcparms +EMPNO(A0050): <employee number>

where:

```
<server address>=supplied IP address
```

ppp=partition

iii=your course assigned ID

<employee number>=valid employee number such as A1002

Notice that your browser window displays the employee information exactly as it did in the previous exercise.



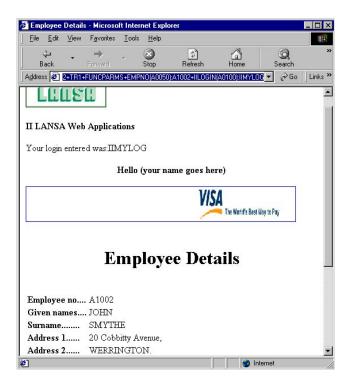
You will now add the iiiLOGIN parameter to the command line entered in Step 1. You will add the following to the URL:

+iiiLOGIN(A0100):iiiMYLOG

so that the URL might now appear as:

```
http://<server address>/cgi-bin/lansaweb?
procfun+iiiproc03+iiifn012+DEM+funcparms +EMPNO(A0050):A1002+iiiL(
```

4. Notice you should now see your iiiMYLOG header component displayed:



### Step 11. Create Page Component for Function iiiFN12 (Optional)

In this step, you will create a page component for the HTML of function iiiFN12. The page component will allow you to easily manipulate the appearance of iiiFN12 and it will prevent changes to the HTML from being replaced when the function is recompiled. You will also change the HTML to display the Employee photo.

- 1. Using the LANSA for the Web Editor, open a new document.
- 2. Delete the default HTML which appears in the new page.
- 3. Open the iiiPROC03 iiiFN120001 HTML page.
- 4. Copy all the HTML statements in the table definition, i.e. copy all code between the following tags

<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3> <TBODY>

copy all HTML code here but do not include the table tags shown above and be

</TBODY> </TABLE>

5. Paste the HTML code into your new page. Your code should appear as follows:

```
<TR>
</TR>
</TR>
</TR>
</TD>
</TD>
</TD>
</TD>
</TD>
</TD>
</TR>
</TD>
</TR>
</TD>
</TR>
</TD>
</TR>
</TD>
</TR>
</TD>
</TR>
</TD>
<
```

</TD> <TD><RDML MERGE="SURNAME "></TD> </TR>< TR ><TD><STRONG><RDML MERGE="@T0004+0046+0060"></STRONG> </TD><TD><RDML MERGE="ADDRESS1 "></TD> </TR>  $\langle TR \rangle$ <TD><STRONG><RDML MERGE="@T0005+0061+0075"></STRONG> </TD> <TD><RDML MERGE="ADDRESS2 "></TD> </TR>  $\langle TR \rangle$ <TD><STRONG><RDML MERGE="@T0006+0076+0090"></STRONG> </TD> <TD><RDML MERGE="ADDRESS3 "></TD> </TR>< TR ><TD><STRONG><RDML MERGE="@T0007+0091+0105"></STRONG> </TD> <TD><RDML MERGE="POSTCODE "EDITCODEO="3"></TD> </TR>  $\langle TR \rangle$ <TD><STRONG><RDML MERGE="@T0008+0106+0120"></STRONG> </TD> <TD><RDML MERGE="PHONEHME "></TD> </TR> $\langle TR \rangle$ <TD><STRONG><RDML MERGE="@T0009+0121+0135"></STRONG> </TD> <TD><RDML MERGE="SALARY "EDITCODEO="1"></TD> </TR>  $\langle TR \rangle$ <TD><STRONG><RDML MERGE="@T0010+0136+0150"></STRONG> </TD> <TD><RDML MERGE="DEPTMENT "></TD> </TR>  $\langle TR \rangle$ 

```
<TD><STRONG><RDML MERGE="@T0011+0151+0165"></STRONG>
</TD>
<TD><RDML MERGE="SECTION "></TD>
</TR>
<TD><STRONG><RDML MERGE="@T0012+0166+0180"></STRONG>
</TD>
</TD>
<TD><RDML MERGE="PHONEBUS "></TD>
</TD>
```

6. Delete all of the field labels whose format is:

```
<<RDML MERGE="@T...">
```

and replace the labels with field descriptions.

Also add an <IMG SRC= > tag to display the PHONEBUS as an image.

The final pages might appear as follows(changes are in bold):

```
< TR >
</TR>
< TR >
<TD><STRONG>Employee Number</STRONG></TD>
<TD><RDML MERGE="EMPNO"
                              "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>First Name</STRONG></TD>
<TD><RDML MERGE="GIVENAME "></TD>
</TR>
<TR>
<TD><STRONG>Last Name</STRONG></TD>
<TD><RDML MERGE="SURNAME "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Address Line 1</STRONG></TD>
<TD><RDML MERGE="ADDRESS1 "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>City</STRONG></TD>
<TD><RDML MERGE="ADDRESS2 "></TD>
```

```
</TR>
< TR >
<TD><STRONG>State</STRONG></TD>
<TD><RDML MERGE="ADDRESS3 "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Postal Code</STRONG></TD>
<TD><RDML MERGE="POSTCODE "EDITCODEO="3"></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Home Phone</STRONG></TD>
<TD><RDML MERGE="PHONEHME "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Salary</STRONG></TD>
<TD><RDML MERGE="SALARY "EDITCODEO="1"></TD>
</TR>
< TR >
<TD><STRONG>Department</STRONG></TD>
<TD><RDML MERGE="DEPTMENT "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Section</STRONG></TD>
<TD><RDML MERGE="SECTION "></TD>
</TR>
\langle TR \rangle
<TD><STRONG>Employee Image</STRONG></TD>
<TD><IMG SRC="/images/<RDML MERGE="PHONEBUS ">"></TD>
</TR>
```

REMINDER: Add the <IMG SRC=...> tag to display the image in the PHONEBUS field.

- 7. Save the document as iiiFN12P with description iiiFN12 Page Component.
- 8. Using the LANSA for the Web editor, open the Maintain Components window.
- 9. Select Add, to add a new component as follows (where iii is your course assigned ID):

Component	iiiFN12P
Туре	Page

Description	iiiFN12 Page Component
Page	iiiFN12P
Mode	OUTPUT

- 10. Close the Maintain Components window.
- 11. Working with your iiiPROC03 process, edit the function named iiiFN12 Employee Details.
- 12. Define a working field named iiiFN12P to be used with the output Web page component and insert a REQUEST command prior to the existing REQUEST command in the function. The new REQUEST will display only one field iiiFN12P with attributes of \*OUTPUT and \*NOID.

Your finished code might appear as follows (where iii is your course assigned ID):

```
FUNCTION (*DIRECT *WEBEVENT)
DEFINE FIELD(#iiiFN12P) TYPE(*CHAR) LENGTH(1)
GROUP_BY
NAME(#EMPLOYEE) FIELDS((#EMPNO *OUTPUT) (#GIVENAME *OU
FETCH
FIELDS(#EMPLOYEE)) FROM_FILE(PSLMST) WITH_KEY(#EMPNO) IC
REQUEST
FIELDS((#iiiFN12P *OUTPUT *NOID)) EXIT_KEY(*NO) MENU_KEY(*N
REQUEST FIELDS(#EMPLOYEE) EXIT_KEY(*NO) MENU_KEY(*NO)
```

- 13. Exit and save your RDML function.
- 14. Compile your RDML function iiiPROC03/iiiFN12.

### Step 12. Test Changes (Optional)

- 1. Execute your function iiiPROC04/iiiFN10. Login using IIIMYLOG
- 2. Select the Administration department and press the SEARCH button. Notice that the *Click Here* button appears exactly as it did in the previous exercise.
- 3. Press the *Click Here* button for the appropriate Section which contains the employee images.
- 4. Click on the employee image.
- 5. A new window should appear showing the employee details:



#### Summary

#### **Important Observations**

- There are many techniques for linking or calling functions. This exercise showed two different examples. The first example used the existing HandleEvent JavaScript function used in LANSA for the Web. In order to pass parameters to the called function, the cprocess>\_HIDDEN fields were used.
- JavaScript requires that a field be on the page in order to change its value. The fields iiiDEPTWK and iiiSECTWK were added to our \_HIDDEN for this purpose.
- When you use an <A HREF> to execute a function, no data is exchanged to the called WEBEVENT function. (The <A HREF> starts a new Web job, as it is calling a new function.) It is exactly the same as calling a function for the first time from a new browser window. The function parameters (funcparms) had to be used to overcome this limitation. In this example, the Employee Number was passed and the called function used this value to fetch the required data.
- In the image example, the image uses the default image directory of your Data/Application Server. You could place these images in any directory and simply add the directory to either the component or the field where the image value is stored in the file.
- When you execute a function from the browser command line, you must pass the parameters you wish the function to use. Parameters are not passed automatically. You added the iiiLOGIN parameter as a funcparm so that the STDHEADER displayed the proper component.
- If you wish to use the iiiLOGIN component, you must use the iiiLOGIN field somewhere in the function as well as passing the parameter to the function. In Step 8, you included a GROUP\_BY statement including iiiLOGIN. You could also put the field in a second request or an IF statement that does nothing. If iiiLOGIN is not defined somewhere in the function, the value cannot be used in the STDHEADER.

### **Tips & Techniques**

• Always create a working field if there is a possibility of more than one function using the field. We could have used the DEPTMENT and SECTION fields, however they will likely be required by other functions,

and if the values are hidden this could cause some problems. These problems are avoided by creating the temporary holding fields. A good Web site layout will ensure that you know where this needs to be done.

- JavaScript can be difficult to debug. There are free or shared software programs available that can make this job easier.
- Creating a component style button is one way of including a link in a browse list. You can also use images or text as links.
- Using HandleEvent is one way of creating a link inside an LANSA function. Another technique is to use the <A HREF> and to add your funcparms to this line, using <RDML MERGE="<fieldname>">.
- In this example, the employee image file names were placed in the PHONEBUS field. If you have an existing database, you can attempt to use an existing field, however, a better technique is to create a parallel file for storing the Web related data. For example, you could create a PLSMT2 file with the same key (EMPNO) but with fields for the images, email addresses, etc.
- If possible, avoid using <A HREF> tags to call separate WEBEVENT functions, as it does not pass any hidden values with it. Instead, modify the HandleEvent JavaScript or review the example given in the online documentation about passing parameters in LANSA for the Web.
- In this example, a new window was opened when using the <A HREF> to call the next function. This is a good technique as it emphasizes the fact that there is no data being passed.

#### What I Should Know

- How to add a link to a browse list.
- How to create process-specific JavaScript to be used with the DEFAULT\_SCRIPT.
- How to pass function parameters within a LANSA function.
- How to build a browse list using the parameters passed from another function.
- How to add an image to a browse list using a page component.
- How to pass function parameters to a LANSA function using funcparms.

### WEB014 - Browse Lists

### **Objective:**

- In this exercise, you will learn how to customize the browse list used in the Employee List function. In the Web page substitution exercises, you learned how to protect your HTML by using Web page components. Using similar techniques, you can learn how to customize the presentation of browse lists.
- To demonstrate how to customize a browse list's appearance.
- To demonstrate the various manipulation techniques associated with browse lists.

To achieve these objectives, you must complete the following:

- Step 1. Create Page Component for Function iiiFN11
- Step 2. Test Changes
- Step 3. Create Graphic Variable for Browse List
- Step 4. Test Changes
- Step 5. Customize Browse List
- Step 6. Test Changes
- Step 7. Add Table Tags Around Browse List
- Step 8. Test Changes
- Step 9. Add Column Headings To Browse List
- Step 10. Test Your Functions
- Summary

### Step 1. Create Page Component for Function iiiFN11

In this step, you will create a page component for the iiiFN11 Employee List function which contains a browse list. The page component will allow you to easily manipulate the appearance of the browse list and it will prevent changes to the HTML from being replaced when the function is recompiled.

- 1. Using the LANSA for the Web Editor, open a new document.
- 2. Delete the default HTML which appears in the new page.
- 3. Open the iiiPROC03 iiiFN11001 HTML page.

Copy all the HTML statements in the table definition, i.e. copy all code between the following tags:

<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3> <TBODY>

copy all HTML code here including the tag below, but do not include the table

<BR>

<RDML MERGE="@BLiiiEMPLST ">

4.Paste the HTML code into your new page.

Your code should appear as follows:

```
<TR>
</TR>
</TR>
</TR>
</TR>
</TD>
</TD>
</RDML MERGE="@T0021+0323+0337"></STRONG>
</TD>
</RDML ONMODE="DIS">
</TD>
</RDML MERGE="iiiDEPTWK "></TD>
</RDML>
</RDML>
</RDML ONMODE="DLT">
</TD>
</RDML MERGE="iiiDEPTWK "></TD>
</RDML>
</RDML MERGE="iiiDEPTWK "></TD>
</RDML>
</RDML>
</RDML>
</RDML ONMODE="ADD">
</RDML ONMODE="ADD"
</RDML ONMODE=
```

```
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
<TD>
<INPUT NAME="AiiiDEPTWK" TYPE="TEXT" SIZE="004" MAXLENGT
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
</TR>
\langle TR \rangle
<TD><STRONG><RDML MERGE="@T0022+0338+0352"></STRONG>
</TD>
<RDML ONMODE="DIS">
<TD><RDML MERGE="iiiSECTWK "></TD>
</RDML>
<RDML ONMODE="DLT">
<TD><RDML MERGE="iiiSECTWK "></TD>
</RDML>
<RDML ONMODE="ADD">
\langle TD \rangle
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
<TD>
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
</TR>
</TBODY>
</TABLE>
</FONT>
\langle BR \rangle
```

<RDML MERGE="@BLiiiEMPLST ">

5. Manually type the names of the iiiDEPTWK and iiiSECTWK fields instead of using the <RDML MERGE="@T....." tags.

The final pages should appear as follows(changes are bold):

```
\langle TR \rangle
</TR>
\langle TR \rangle
<TD><STRONG>Department</STRONG></TD>
<RDML ONMODE="DIS">
<TD><RDML MERGE="iiiDEPTWK "></TD>
</RDML>
<RDML ONMODE="DLT">
<TD><RDML MERGE="iiiDEPTWK "></TD>
</RDML>
<RDML ONMODE="ADD">
\langle TD \rangle
<INPUT NAME="AiiiDEPTWK" TYPE="TEXT" SIZE="004" MAXLENGT
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
\langle TD \rangle
<INPUT NAME="AiiiDEPTWK" TYPE="TEXT" SIZE="004" MAXLENGT
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
</TR>
< TR >
<TD><STRONG>Section</STRONG></TD>
<RDML ONMODE="DIS">
<TD><RDML MERGE="iiiSECTWK "></TD>
</RDML>
<RDML ONMODE="DLT">
<TD><RDML MERGE="iiiSECTWK "></TD>
</RDML>
<RDML ONMODE="ADD">
<TD>
```

```
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
<TD>
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
</RDML>
</TR>
```

</TBODY> </TABLE> </FONT>

<BR>

```
<RDML MERGE="@BLiiiEMPLST ">
```

- 6. Save the document as iiiFN11P with description iiiFN11 Page Component.
- 7. Using the LANSA for the Web editor, open the Maintain Components window.
- 8. Select Add, to add a new component as follows (where iii is your course assigned ID):

Component	iiiFN11P
Туре	Page

Description	iiiFN11 Page Component
Page	iiiFN11P
Mode	OUTPUT

- 9. Close the Maintain Components window.
- 10. Working with your iiiPROC03 process, edit the function named iiiFN11 Employee List.

Define a new field iiiFN11P which will be used for the Web page component iiiFN11P.

You will manually insert a Display command prior to the display command that is already in this function. It will display only one field iiiFN11P with an attribute of \*NOID.

The finished code might appear as follows:

```
FUNCTION (*DIRECT *WEBEVENT)
       FIELD(#iiiIMAGE) REFFLD(#PHONEBUS)
DEFINE
        FIELD(#iiiFN11P) TYPE(*CHAR) LENGTH(1)
DEFINE
DEF_LIST NAME(#iiiEMPLST) FIELDS((#iiiIMAGE *NOID)(#EMPNO)
(#GIVENAME) (#SURNAME)(#PHONEBUS *HIDDEN))
CLR_LIST NAMED(#iiiEMPLST)
SELECT
FIELDS((#iiiEMPLST)) FROM FILE(PSLMST1) WITH KEY(#iiiDEPTWK
ADD_ENTRY TO_LIST(#iiiEMPLST)
ENDSELECT
DISPLAY
FIELDS((#iiiFN11P *NOID)) EXIT KEY(*NO) MENU KEY(*NO)
DISPLAY
FIELDS((#iiiDEPTWK) (#iiiSECTWK)) BROWSELIST(#iiiEMPLST) EXIT
```

- 11. Exit and save your RDML function.
- 12. Compile your RDML function iiiPROC03/iiiFN11.

### **Step 2. Test Changes**

You may wish to test your changes before continuing. It is always a good idea to ensure that the function displays properly after creating a component page.

- 1. Execute your function iiiPROC04/iiiFN10.
- 2. Login using iiiMYLOG.
- 3. Select the Administration department and press the SEARCH button.

Notice that the *Click Here* button appears exactly as it did in the previous exercise.

4. Press the *Click Here* button for the appropriate Section containing the employee images.

The iiiFN11 should appear as before, except that it is now using the iiiFN11P Web page component.

#### **Step 3. Create Graphic Variable for Browse List**

In this step, you will create a graphic variable for your browse list in iiiFN11. The LANSA for the Web Editor will be used to manually create the graphic variable. The Web Functions Wizard could also be used to configure the browse list.

- 1. Using the LANSA for the Web editor, open the Graphic Variables window.
- 2. Click the Add button.
- 3. Create a new Graphic Variable as follows (where iii is your course assigned ID):
  - Name: **\*LW3BL\_iiiEMPLST**
  - Type: **TEXT**
  - Value: \*NOBPCIT
- 4. Click OK. Close the Graphic Variables window.

## **Step 4. Test Changes**

In this step, you will test the changes made to the browse list by the graphical variable.

- 1. Execute your function iiiPROC04/iiiFN10.
- 2. Login using iiiMYLOG.
- 3. Select the Administration department and press the SEARCH button.

Notice that the *Click Here* button appears exactly as it did in the previous exercise.

4. Press the *Click Here* button for the appropriate Section which contains the employee images.



You will notice that the layout of the information is not well organized, as your graphic variable specified no table tags. (The browse list appearance will be improved in the upcoming steps.)

## **Step 5. Customize Browse List**

In this step, you will create a page component for the browse list used in function iiiFN11. This component will allow you to customize the browse list appearance. After adding the component to the browse list, all other fields will be hidden so that only the component appears on your Web page. This technique is the best way to manipulate the layout of a browse list without having recompiles affect the changes.

- 1. Using the LANSA for the Web editor, create a new HTML page.
- 2. When the new page appears, delete all of the default HTML code.
- 3. Add the following lines to the page:

<TR>

<TD VALIGN="MIDDLE"><RDML COMPONENT="iiiIMAGE"></TD> <TD VALIGN="MIDDLE"><RDML MERGE="GIVENAME"></TD> <TD VALIGN="MIDDLE"><RDML MERGE="SURNAME"></TD> <TD VALIGN="MIDDLE"><RDML MERGE="EMPNO"></TD> </TR>

Notice that you have changed the order of the fields from the order used in the browse list.

- 4. Save the document as iiiBL11 with description iiiFN11 Browse List Component.
- 5. Using the LANSA for the Web editor, open the Maintain Components window.
- 6. Select Add, to add a new component as follows (where iii is your course assigned ID):

Component iiiBL11

Type Page

Description iiiFN11 Browse List Component

Page iiiBL11

Mode OUTPUT

- 7. Close the Maintain Components window.
- 8. Working with your iiiPROC03 process, edit the function named iiiFN11 Employee List.

Define a new field iiiBL11 which will be used for the Web page component iiiBL11.

DEFINE FIELD(#iiiBL11) TYPE(\*CHAR) LENGTH(1)

Change the browse list by removing the iiiIMAGE field as this component is now embedded using the iiiBL11 Web component. Change the remaining fields to use an attribute of \*HIDDEN. Finally, add the field iiiBL11 with an attribute of \*NOID to the beginning of the list.

The DEF\_LIST should appear as follows:

```
DEF_LIST NAME(#iiiEMPLST) FIELDS((#iiiBL11 *NOID)
(#EMPNO *HIDDEN) (#GIVENAME *HIDDEN) (#SURNAME *HIDDEN)
(#PHONEBUS *HIDDEN)) ENTRYS(9999)
```

9. Exit and save your RDML function.

10. Compile your RDML function iiiPROC03/iiiFN11.

### Step 6. Test Changes

Execute your function iiiPROC04/iiiFN10.

- 2. Login using iiiMYLOG.
- 3. Select the Administration department and press the SEARCH button. Notice that the *Click Here* button appears exactly as it did in the previous exercise.
- 3. Press the *Click Here* button for the appropriate Section which contains the employee images:



Your iiiFN01 browse list should be displayed as before, except that the fields being displayed are controlled by the iiiBL11 Web page component which is substituted for the <RDML MERGE="@BLiiiEMPLST "> in the iiiFN11P. If you view the source for the page, you will notice that the <TR> and <TD> tags are being sent to the browser (as part of the iiiBL component); however, the <TABLE> tags have not be added so the data is not formatted. The <TABLE> tags will be added in the next step.

Notice that the EMPNO field is now displayed as the last field based on the order of the table data you specified in the iiiBL11 Web page component.

## Step 7. Add Table Tags Around Browse List

In this step, you will correct the appearance of the browse list on the employee list screen. You will do this by adding table tags around the browse list tag inside of the HTML component iiiFN11P.

- 1. Using the LANSA for the Web Editor, open the HTML page iiiFN11P.
- 2. Edit the page by placing the following open and close table tag around the browse list tag. \*Hint: This tag can be recognized by searching for <RDML MERGE="@.

```
<TABLE BORDER="1" cellpadding="10" cellspacing="1">
</TABLE>
```

3. Your page should contain the following code:

•••

</TBODY> </TABLE> </FONT>

<BR>

```
<TABLE BORDER="1" cellpadding="10" cellspacing="1">
<RDML MERGE="@BLiiiEMPLST">
</TABLE>
```

4. Save the document.

#### **Step 8. Test Changes**

Execute your function iiiPROC04/iiiFN10.

- a. Login using iiiMYLOG.
- b. Select the Administration department and press the SEARCH button.

Notice that the *Click Here* button appears exactly as it did in the previous exercise.

3. Press the *Click Here* button for the appropriate Section which contains the employee images.



## Step 9. Add Column Headings To Browse List

In this step, you will add column headings to your table to give the data in your browse list meaning. This will also be done in the page component iiiFN11P.

- 1. Using the LANSA for the Web editor, open the HTML page iiiFN11P.
- 2. Edit the page by placing the following code between the <TABLE> tag you just added in the step above and the <RDML MERGE="@BLiiiEMPLST"> line.

```
<TR>
```

```
<TD ALIGN="LEFT">Employee <BR>Photo</TD>
<TD COLSPAN="2" ALIGN="LEFT">Employee <BR>Name</TD>
<TD ALIGN="LEFT">Employee<BR> Number</TD>
</TR>
```

3. The final HTML code for iiiFN11P should appear as follows:

```
\langle TR \rangle
</TR>
\langle TR \rangle
<TD><STRONG>DEPARTMENT:</STRONG></TD>
<RDML ONMODE="DIS">
<TD><RDML MERGE="DEPTMENT "></TD>
</RDML>
<RDML ONMODE="DLT">
<TD><RDML MERGE="iiiDEPTWK "></TD>
</RDML>
<RDML ONMODE="ADD">
<TD>
<INPUT NAME="AiiiDEPTWK" TYPE="TEXT" SIZE="004" MAXLENGT
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
\langle TD \rangle
<INPUT NAME="AiiiDEPTWK" TYPE="TEXT" SIZE="004" MAXLENGT
VALUE="<RDML MERGE="iiiDEPTWK ">"
onFocus="SetNameLocation('AiiiDEPTWK',03,18)"></TD>
</RDML>
```

```
</TR>
\langle TR \rangle
<TD><STRONG>SECTION:</STRONG></TD>
<RDML ONMODE="DIS">
<TD><RDML MERGE="SECTION"></TD>
</RDML>
<RDML ONMODE="DLT">
<TD><RDML MERGE="iiiSECTWK "></TD>
</RDML>
<RDML ONMODE="ADD">
\langle TD \rangle
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
<RDML ONMODE="CHG">
< TD >
<INPUT NAME="AiiiSECTWK" TYPE="TEXT" SIZE="002" MAXLENGT
VALUE="<RDML MERGE="iiiSECTWK ">"
onFocus="SetNameLocation('AiiiSECTWK',04,18)"></TD>
</RDML>
</TR>
</TBODY>
</TABLE>
</FONT>
```

```
<BR>
<TABLE BORDER="1" cellpadding="10" cellspacing="1">
<TR>
<TD ALIGN="LEFT">Employee <BR>Photo</TD>
```

```
<TD COLSPAN="2" ALIGN="LEFT">Employee <BR>Name</TD>
<TD ALIGN="LEFT">Employee<BR> Number</TD>
</TR>
<RDML MERGE="@BLiiiEMPLST ">
</TABLE>
```

#### 4. Save the document.

#### **Step 10. Test Your Functions**

In this step, you will test the changes made above, as this will ensure that the changes to your function iiiFN12 have occurred. It is also a good idea to ensure that the function displays after creating a component page.

- 1. Execute your function iiiPROC04/iiiFN10.
- 2. Login using iiiMYLOG.
- 3. Select the Administration department and press the SEARCH button.

Notice that the *Click Here* button appears exactly as it did in the previous exercise.

3. Press the *Click Here* button for the appropriate Section containing the employee images.



#### Summary

### **Important Observations**

- Creating a page component is one way of enhancing the appearance of any page. You can use them stand alone or inside other components.
- The <RDML MERGE="@BLiiiEMPLST "> tag is very important when customizing a browse list. LANSA dynamically generates the browse list HTML so you cannot edit it directly. By using Web page components, and by knowing the list content, you can control the browse list presentation.
- The graphical variable\*LW3BL\_iiiEMPLST is very important. The \*NOBPCIT settings allow you to customize the browse list presentation as described in Step 7 and 9.
- The iiiBL11 Web page component allows you to control the browse list presentation. You can add images (as in the case of the iiiIMAGE component) or you can change the order that the information is presented. It is very important to note that the iiiIMAGE component works because PHONEBUS is a hidden field in the browse list. PHONEBUS contains the data required by the iiiIMAGE component.
- The use of the two DISPLAY statements is very important to this technique. Only the first DISPLAY is executed when the function is called. However, the data defined in the browse list of the second DISPLAY will be exchanged with the function.

DISPLAY

FIELDS((#iiiFN11P \*NOID)) EXIT\_KEY(\*NO) MENU\_KEY(\*NO) DISPLAY

FIELDS((#iiiDEPTWK) (#iiiSECTWK)) BROWSELIST(#iiiEMPLST) EXIT

• Once the iiiBL11 Web page component was inserted into the browse list, the browse list definition was altered so that all fields were made to be hidden and the iiiIMAGE component was removed. The iiiIMAGE field was included directly in the iiiBL11 component.

DEF\_LIST NAME(#iiiEMPLST) FIELDS((#iiiBL11 \*NOID) (#EMPNO \*HIDDEN) (#GIVENAME \*HIDDEN) (#SURNAME \*HIDDEN) (#PHONEBUS \*HIDDEN)) ENTRYS(9999)

# **Tips & Techniques**

• The browse list customization technique described in this exercise is a very

common method of enhancing the presentation of your Web functions.

# What I Should Know

- How to build a browse list inside a Web page component.
- How to customize a browse list.

### WEB015 - Data Apportionment

#### **Objective:**

- In this exercise, you will learn how data apportionment is performed with LANSA for the Web. You will create a single screen which allows the user to enter some comments into an input text area in the browser. When the screen is submitted, the entered text will be divided into a group of working fields. The working fields will be redisplayed on the screen, just below the input text area.
- To learn how to overcome the 256 character field limitation for input fields used with text areas in the browser.
- To learn how to create a page component containing both input and output components using only one REQUEST.

To achieve these objectives, you must complete the following:

- Step 1. Create iiiFN13 Contact Us
- Step 2. Create a New Page Component
- Step 3. Compile and Test Your Function iiiFN13
- Step 4. Using JavaScript Alerts (Optional)
- Summary

# Step 1. Create iiiFN13 – Contact Us

In this step, you will create function iiiFN13 Contact Us in process iiiPROC03. This function will simply request data using an input text area and then it will display the contents of the fields after the data apportionment has taken place.

- 1. Working with your iiiPROC03 process, create a new function named iiiFN13 Contact Us. You will manually enter the code for the function.
- 2. Write the RDML Code to perform the following tasks:
- Define a field called iiiCOMENT to be used for Web page component substitution.
- Define three fields named COMMNT, COMMNT001, and COMMNT002. These fields will be used as the input fields from the text area. Fields are alpha, length 20.
- Define three fields named COMOUT, COMOUT001, COMOUT002. These fields will be used to display the values of the COMMNT fields.
- Change the values of COMOUT, COMOUT001, AND COMOUT003 to the values of COMMNT, COMMNT001 AND COMMNT002.
- Change COMMNT, COMMNT001, COMMNT002 fields to \*BLANKS.
- Request the page component iiiCOMENT. Include a SUBMIT user key which will call iiiFN13 (re-entrant WEBEVENT function).

#### Solution:

One possible solution to this exercise is shown below (where iii is your course assigned ID):

```
FUNCTION (*DIRECT *WEBEVENT)
       FIELD(#iiiCOMENT) TYPE(*CHAR) LENGTH(1)
DEFINE
       FIELD(#COMMNT) TYPE(*CHAR) LENGTH(20)
DEFINE
DEFINE
        FIELD(#COMMNT001) TYPE(*CHAR) LENGTH(20)
        FIELD(#COMMNT002) TYPE(*CHAR) LENGTH(20)
DEFINE
        FIELD(#COMOUT) TYPE(*CHAR) LENGTH(20)
DEFINE
        FIELD(#COMOUT001) TYPE(*CHAR) LENGTH(20)
DEFINE
        FIELD(#COMOUT002) TYPE(*CHAR) LENGTH(20)
DEFINE
         FIELD(#COMOUT) TO(#COMMNT)
CHANGE
         FIELD(#COMOUT001) TO(#COMMNT001)
CHANGE
         FIELD(#COMOUT002) TO(#COMMNT002)
CHANGE
CHANGE
```

FIELD(#COMMNT #COMMNT001 #COMMNT002) TO(\*BLANKS) REQUEST

FIELDS((#iiiCOMENT \*NOID)) EXIT\_KEY(\*NO) MENU\_KEY(\*NO) USE

- 3. Exit and save your RDML function.
- 4. Do not compile the function at this time.

### Step 2. Create a New Page Component

In this step, you will create the new page component iiiCOMENT that will be used with the function you created in Step 1. It will be a very simple page containing one input text area (3 rows by 20 characters wide) and three output fields to display the contents of the text box. You will also register your key word SUBMIT to create a WEBEVENT link so that iiiFN13 calls itself.

1. Using the LANSA for the Web editor, create a new HTML page.

- 2. When the new page appears, delete all the default HTML code.
- 3. Add the following lines to the page:

```
<STRONG>Please enter your question below:</STRONG>
<BR>
<RDML CHECKVALUE="YES">
<TEXTAREA NAME="COMMNT -
L020" WRAP="PHYSICAL" ROWS=3 COLS=20>
<RDML MERGE="COMMNT">
<RDML MERGE="COMMNT">
<RDML MERGE="COMMNT001">
</TEXTAREA>
</RDML>
```

<P>

```
<STRONG>Your text is stored as:</STRONG>
<BR>
COMMNT field:  <RDML MERGE="COMOUT ">
<BR>
COMMNT001 field:  <RDML MERGE="COMOUT001">
<BR>
COMMNT002 field:  <RDML MERGE="COMOUT002">
```

**Important Note:** Be sure to enter the text area command exactly as show. There should be 4 blank spaces between COMMNT and -L020.

#### <TEXTAREA NAME="COMMNT -L020" WRAP="PHYSICAL" ROWS=3 COLS=20>

4. Save your document as iiiCOMENT with description Data Apportionment

Component.

- 5. Using the LANSA for the Web editor, open the Maintain Components window.
- 6. Select Add, to add a new component as follows. (where iii is your course assigned ID):

Component	iiiCOMENT
Туре	PAGE

Description	Data Apportionment Component
Page	iiiCOMENT
Mode	Input

- 7. Close the Maintain Components window.
- 8. Use the Tools menu category and select the Keywords Maintain option. Do not enter a process. Press the OK button to continue. Press the add button to create a link to by entering the following information:

Process	iiiPROC03
Function	iiiFN13
Keyword	SUBMIT
Description	Submit Request
Linked Process	iiiPROC03
Linked Function	iiiFN13

## Step 3. Compile and Test Your Function iiiFN13

In this step, you will compile your function iiiFN13 so that the new component is recognized. Remember that without the recompile you will not see the component you just created.

- 1. Compile your function iiiPROC03/iiiFN13.
- 2. Check that your compile completed successfully.
- 3. Execute your function iiiPROC03/iiiFN13.
- 4. Enter any information you like into the text box in less than 60 characters.
- 5. Press the Submit Request button to view the output.



6. Enter more than 60 characters.

Notice what happens to the data when you click the Submit Request button.

Back	Forward	Stop	Refresh	Home	S
Address 🖉 http	p://192.168.33.95/d	cgi-bin/lansaweb	?procfun+iiproc03	3+iifn013+tr1	•
Your login er	ntered was:				
			5	11	
	Summer Shares	ar Bards			
		<b>a</b> .			
		Cont	act us		
		Cont	act us		
Please ente	er your questio		act us		
Please ente	er your questio		act us		
Enter your te	xt here -		act us		
	xt here -		act us		
Enter your te in less than 6	xt here D characters		act us		
Enter your te in less than 6 Your text is	xt here O characters		act us		
Enter your te in less than 6	xt here 0 characters 5 <b>stored as:</b> FIELD:		act us		

The error message is displayed:

Microsoft Internet Explorer
You have exceeded the maximum size for this field.
OK

# Step 4. Using JavaScript Alerts (Optional)

In this step, you will add JavaScript to your component iiiCOMENT to ensure that the user cannot add anymore text than is available.

Using the LANSA HTML Editor, open your iiiCOMENT page.

Add the following code to the HTML for the text area:

```
onkeyup="ChkLngTBox(this,60);"
```

The code should now appear something like below.

```
<RDML CHECKVALUE="YES">
<TEXTAREA NAME="COMMNT -
L020" WRAP="PHYSICAL" ROWS="3" COLS="20" onkeyup="ChkLngTB(
<RDML MERGE="COMMNT">
<RDML MERGE="COMMNT001">
<RDML MERGE="COMMNT002">
</TEXTAREA>
</RDML>
```

3. Add the following code to the very bottom of the HTML for iiiCOMENT:

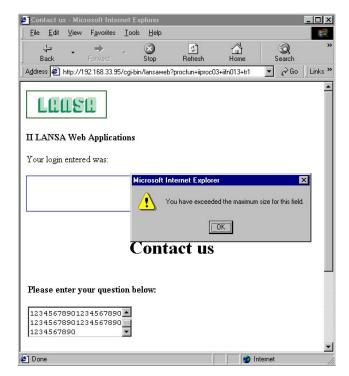
```
<SCRIPT LANGUAGE="JavaScript">
function ChkLngTBox(object,mylength)
{
var x=object.value
var mylength
if (x.length >= ++mylength){
alert('You have exceeded the maximum size for this comment field.');
x=x.substr(0,mylength);
object.value=x;
}
</SCRIPT>
```

- 4. Save your page iiiCOMENT.
- 5. Execute your function iiiPROC03/iiiFN013 and attempt to key in more than

60 characters.

Notice that no compile is necessary for this change to occur.

A message should appear, telling you that you have exceeded the maximum size for this field. If it does not, check that there is not a script error or that you have mistyped a variable name. (Remember: JavaScript is a case sensitive language.



## Summary

## **Important Observations**

- In this example, the data apportionment fields were only defined as 20 characters. In your applications, your fields could be much larger (up to 256 characters).
- Data apportionment provides a solution to the 256 character field limitation by parsing the data into a grouping of related fields. A user could enter 500 characters of text which cannot be stored in a single LANSA field. Using data apportionment, the text can be stored in two fields.
- In this example, the fields were simply written back out to the screen to show you how the data apportionment is performed. In your application, these fields might be written to a database file.

# **Tips & Techniques**

- The fields used for the data apportionment must be exactly 9 characters in length. If the primary field name is EXAMPLE, then the related field names would be EXAMPLE01 and EXAMPLE02, etc.
- When specifying the length of the data apportionment (default length is 256 characters), the "–Lnnn" text specification must start in the eleventh position, i.e. the field name must allow for 10 characters (9 character field name plus a blank). For example:
- <TEXTAREA NAME="COMMNT -L020" WRAP="PHYSICAL" ROWS=3 COLS=20>
- The column width of the text area does not have to equal the length of the apportionment fields, however it does help to determine if the text area apportionment is working correctly. In this example, the text area could have been ROWS=6 COLS=10, or ROWS=2 COLS=30.
- JavaScript can be very helpful in ensuring that the correct data enters your system. Along with the line added to the text area in the above example you could also add onChange="ChkLngTBox(this,60);" after the last quote of the onkeyup. This will also catch an overflow if the user pastes data to the field.

# What I Should Know

- How to add a text area to a LANSA Web page.
- How to use data apportionment to solve the 256 character field limitation.

• How to use data apportionment to enhance Web pages.

# WEB016 - Customizing Personnel Application (Optional)

## **Objective:**

- In this optional exercise, you may use the Web Functions Wizard (described in the Web Functions Wizard Guide to customize your Personnel System application. You could complete this exercise before you begin exercise WEB010 (i.e. before you have started building your application) or after you have completed exercise WEB015 (i.e. after all the functions have been created).
- To customize the color schemas or presentation layout of your Personnel System application.
- To define the menu components for your application.
- To define the user navigation within the application.

To achieve these objectives, you must complete the following:

- Step 1. Design Your Layout
- Step 2. Test
- Summary

## **Step 1. Design Your Layout**

In this step, you will design the general navigation and layout of the application.

- 1. Decide on the general layout you would like to use for your application.
- 2. Determine the menu components that you will require to create the desired user navigation.

You might want to create links to the Login and Contact Us functions using a vertical menu.

Or, you might want to create DHTML menus for the various department and section functions.

- 3. Execute the Web Functions Wizard to build your application interface.
- 4. Tip: Remember that you can adopt layouts between your processes in order to quickly create a common look-and-feel to your application.

## Step 2. Test

1. Execute your test application and verify that the user navigation works as designed.

## Summary

### **Important Observations**

- You can create your layouts before you create your processes and functions. The Web Functions Wizard creates a layout definition which will be used by the processes and functions once they are created. Obviously, if a function has not been created, you cannot test the link to the function.
- Remember, the STDTABFLR field must be included in your functions as a hidden field or it can be added to DEFAULT\_HIDDEN, if you wish the selected function to be highlighted.

# **Tips & Techniques**

- Using the Web Functions Wizard , you can quickly and easily define the layout and user navigation for your application. Try different layouts and schemas.
- Remember to make note of the visualization and color selected, as the Wizard does not show your existing settings.
- Think carefully when specifying the Reuse Job and Start in New Window options. You must understand the type of function executing and the navigation required to and from the function.
- Reminder: If you are using a tool bar, you will not usually select the *Start in New Window* option.
- Do not use the user defined visual style for the Horizontal Menu components.

## What I Should Know

• How to use the Web Functions Wizard to customize your application.

## **Personnel Demonstration System**

For details about the Personnel Demonstration System shipped with the LANSA software, review the following:

Personnel System Demonstration Files

Physical Database Map of Personnel System

Sample Data in the Personnel Files

For details about installing the Personnel Demonstration System, refer to the Tutorials for Web Functions & WEBEVENTs.

## **Personnel System Demonstration Files**

The company has a simple organizational structure. It is divided into departments such as Administration, Audit, Information Services, Legal, Travel, etc. Each of these departments may have one or more sections such as Accounting, Purchasing, Sales, etc. The Department table (DEPTAB) stores the list of departments. The Section table (SECTAB) is used to store the sections within each department.

The Personnel Master file (PSLMST) stores details about each employee. For example, the employee's name, address, and telephone number are stored in this master file. As each employee works in a section of a department, this information is also stored in the Personnel Master file.

Each employee also has a list of skills. For example, an employee might have Cobol, C and C++ programming skills or management and administration skills. A Skills table (SKLTAB) is used to store the skill codes. A Personnel Skills file (PSLSKL) stores the specific skills of each employee.

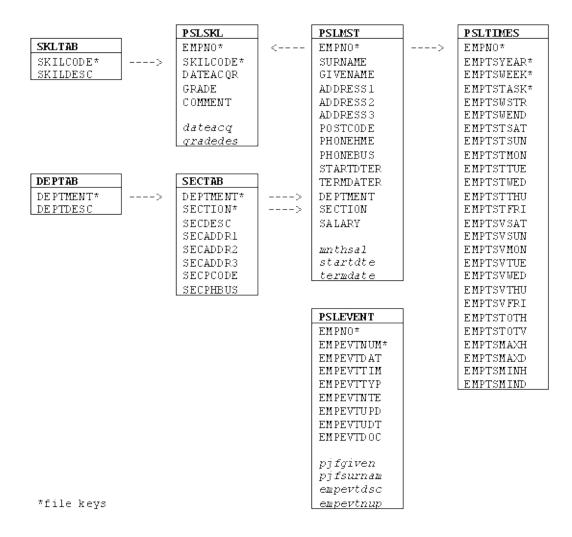
The Personal Event Log file (PSLEVENT) allows significant events and notes to be recorded against an employee. It logically extends the PSLMST file. It is an RDMLX file and therefore will only be available in an RDMLX partition.

The Personnel Time Sheet file (PSLTIMES) records employee time sheet details. Details are recorded by week number (1 to 52) within a year for each employee. It is designed mostly for use with L/Client and to show extensive trigger power by performing relatively complex calculations and storing them in the DBMS without the application needing to know what is happening. Note that all the data is created and stored in the DBMS when information is created or updated, which means that L/Client applications have read access to it without needing to use the triggers. It is an RDMLX file and therefore will only be available in an RDMLX partition. It contains examples of a number of RDMLX field types including BLOB.

The physical database layout follows.

# **Physical Database Map of Personnel System**

(Including Virtual and Predetermined Join Fields)



**Historical note:** The LANSA Personnel Demonstration System was originally created in 1987 to execute on a System 38. This same application can still be executed on IBM i as well as Windows, WEB and Linux platforms. The original database and application are virtually unchanged but there have been some extensions to the database to demonstrate RDMLX concepts.

This shows how LANSA can protect your investment in your applications.

There are two web versions of the Personnel System, one uses WEBEVENT technology and the other uses WAMs. These are included with the IBM i demonstration material and can be checked out to a Visual LANSA environment as required. These applications can be used to give you ideas of how to create and include the elements required for both WEBEVENT and WAM processing. webevent (web enabled RDML or RDMLX partition):
http://<web server>/cgi-bin/lansaweb?
procfun+lansadem+ldem+dem

wam (web enabled RDMLX partition, only supported for technology service LANSA:XHTML): http://<web server>/CGI-BIN/lansaweb? webapp=LWAMDEM+webrtn=ldemhome+ml=LANSA:XHTML+

## **Sample Data in the Personnel Files**

Following is a list of some of the sample data in the Personnel File which may be contained in the files. As developers may edit these files, the data you see may have been different to the following:

<b>DEPTAB:</b>	SECTAB:	<b>PSLMST:</b>
DEPTMENT	DEPTMENT/SECTION	EMPNO
ADM	ADM 01	17 employees
	ADM 02	A1002
		A1005
		A1014
		A8888
	ADM 03	
	ADM 04	
	ADM 05	
AUD	AUD 01	
	AUD 02	
	AUD 03	
FLT	FLT 01	
	FLT 02	
	FLT 03	
INF	INF 01	
	INF 02	
	INF 03	

# **Appendix A. Header Styles**

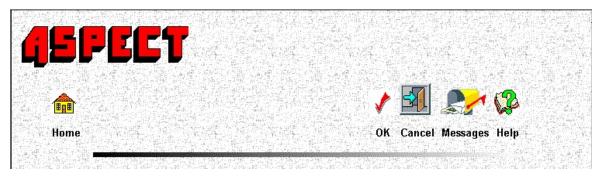
LANSA for the Web provides support for different styles of headers for Web function applications. Generally speaking, the header styles can be either static or dynamic. In addition, the headers can either have a scrolling or non-scrolling attribute.

Static Header Dynamic Header Scrolling Header Non-Scrolling Header Frameset Definition Limitations of non-scrolling headers Combination Of Scrolling And Non Scrolling Header Styles Override Header Style

# **Static Header**

Static headers are used where the buttons for your LANSA functions do not change from one function to another. All the buttons defined in the STDHEADER page are displayed for every function.

To use this header style in your application, copy the STDHEADER\_STYLE1 page to the STDHEADER page in your system.



The standard header consists of:

- a company logo displayed at the top left hand corner.
- a set of buttons (with configurable images) consisting of:

Button	Description
Home	A button, which emulates an Exit key and will take you back to a nominated home page.
Menu	This will take you back to the Process Menu level. This button emulates a Cancel key, until you are returned to the Process Menu level.
OK	Equivalent to the <enter> key.</enter>
Cancel	Equivalent to Cancel (F12).
Messages	Equivalent to Messages (F14).
Help	Equivalent to Help (F1). Help is displayed in a new browser window, with the toolbar and menu bar disabled.

The static header style provided by LANSA for the Web incorporates images for the standard buttons.

Static headers are used where you want to provide a consistent headers throughout your application. However, you will need to be careful with your

functions to ensure that the buttons used in the static header are always enabled. Static headers do not have the capability to check the state of a particular button. Static headers assume that the buttons displayed are always enabled.

If your functions do not have the same set of buttons enabled all the time, you must use the dynamic header style.

To comply with the XHTML 1.0 transitional document type definition, LANSA uses JavaScript functions. JavaScript support must be enabled in your browser.

# **Dynamic Header**

LANSA for the Web provides support for dynamic display of buttons in the Standard Header.



This means that you can define all the buttons used in your applications in the Standard Header component. By encapsulating each of these buttons with the <RDML BUTTON> tag, LANSA for the Web will dynamically enable or disable the particular button, according to its status in your application.

```
<img src="<RDML MERGE="*LW3CPYLOGO">" alt="Logo" /><br />
<RDML BUTTON="&WEBEVENT">
<script type="text/javascript" language="javascript">
//<![CDATA[
function ButtonClick(button)
{
 document.LANSA._BUTTON.value=button; document.LANSA.submit();
}
//]]>
</script>
</RDML>
<table cellpadding="0" cellspacing="0" border="0" width="100%" align="left
<RDML BUTTON="&EXIT">
 <a href="javascript:ButtonClick('HOME');">
<img src="/IMAGES/TB_HOME.GIF" alt="Home" border="0" width="55" he
</a>
</RDML>
<RDML BUTTON="&CANCEL">
 <a href="javascript:ButtonClick('MENU')">
<img src="/IMAGES/TB_MENU.GIF" alt="Menu" border="0" width="55" he
</a>
</RDML>
<RDML BUTTON="&WEBEVENT">
```

```
<img src="/IMAGES/TB_NEXT.GIF" alt="Next" border="0" width="55" heig
</a>
</RDML>
<RDML BUTTON="&CANCEL">
 <a href="javascript:ButtonClick('CANCEL')">
<img src="/IMAGES/TB_BACK.GIF" alt="Back" border="0" width="55" hei
</a>
</RDML>
<RDML BUTTON="&ADD">
 <a href="javascript:ButtonClick('ADD')">
<img src="/IMAGES/TB_ADD.GIF" alt="Add" border="0" width="45" heigh
</a>
</RDML>
<RDML BUTTON="&CHANGE">
 <a href="javascript:ButtonClick('CHANGE')">
<img src="/IMAGES/TB CHG.GIF" alt="Change" border="0" width="65" he
</a>
</RDML>
<RDML BUTTON="&DELETE">
 <a href="javascript:ButtonClick('DELETE')">
<img src="/IMAGES/TB_DLT.GIF" alt="Delete" border="0" width="65" heig
</a>
</RDML>
<RDML BUTTON="&PROMPT">
 <a href="javascript:ButtonClick('PROMPT')">
<img src="/IMAGES/TB_SRCH.GIF" alt="Search" border="0" width="70" he
</a>
</RDML>
<RDML BUTTON="&WEBEVENT">
 <a href="javascript:GetHelp()">
<img src="/IMAGES/TB_HELP.GIF" alt="Help" border="0" width="55" heig
</a>
</RDML>
 <br clear="all" /><br />
```

<a href="javascript:ButtonClick('OK')">

Dynamic headers are very powerful as the buttons shown in the header, when the function is executed, are the buttons that are enabled for the function. This is unlike static headers, where the displayed buttons may not be valid for the function.

For example, in one of your functions, you have disabled the Exit key. This will be detected in dynamic headers and the Exit key (and hence the Home button) will not be displayed. However, in static headers, the Exit key (and hence the Home button) will still be displayed if the Exit key was defined in your static header.

Dynamic headers allow you to define all the buttons used in your application in a single Web component, the STDHEADER component. LANSA for the Web will then determine the status of each button in the component dynamically.

It is recommended that you incorporate the dynamic buttons support provided as the default header style in LANSA Web function applications.

The dynamic header style page shipped with LANSA for the Web uses a Tool Bar to display the buttons. The trend in Web applications is to use Tool Bars instead of images to represent actions.

LANSA for the Web provides you with a collection of images in GIF format. These images are used to build the Tool Bar in the STDHEADER component. These images are prefixed with 'TB\_'. LANSA for the Web also provides you with templates of images that can be used to create your own set of images for the Tool Bar. The images were created using Paint Shop Pro.

This is the default header style in LANSA for the Web. This style page is also contained in the STDHEADER\_STYLE2 page in your system.

If you have changed the header style and you want to restore the default header style, then copy the STDHEADER\_STYLE2 page to be the STDHEADER page in your system.

# **Scrolling Header**

The scrolling header style embeds the STDHEADER component into the HTML generated for the function. The buttons in the component are part of the HTML displayed in your browser. This means that as you use the scroll bars in your browser, you are scrolling through the body of the function, which includes the buttons in your STDHEADER component.

In HTML terminology, your buttons and the body of the function are displayed within the single frame.

This style is not user friendly as your STDHEADER component is usually at the top of the page. If your HTML page displays many fields, the buttons of your function may not be visible if you have scrolled down through the body of your function.

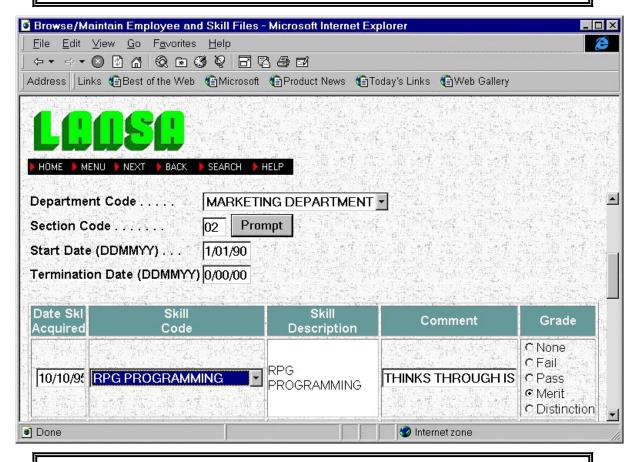
For functions with large display, it would be ideal if you could have the buttons displayed in a static frame and the body of the function in a separate frame. These are the features provided by the non-scrolling style.

To comply with the XHTML 1.0 transitional document type definition, LANSA uses JavaScript functions. JavaScript support must be enabled in your browser.

# **Non-Scrolling Header**

LANSA for the Web also provides you with a non-scrolling header style for Web function applications. With this style, the STDHEADER page is loaded into a separate frame and your LANSA function is loaded into another frame. This feature allows you to scroll through the body of your function, while the buttons remain static. This means that the buttons are available to you all the time.

Before deploying this header style in your application, read the Limitations of non-scrolling headers list to understand the limitations of this header style.



To enable this feature, you must install the STDHEADER\_STYLE3 page as the default STDHEADER page. You can use the Web Function Editor to open the STDHEADER\_STYLE3 page and then save the page as STDHEADER.

Next, you will need to set the Use Non-Scrolling Header option in the **Presentation tab** of the LANSA Administrator described in the *Installing LANSA on Windows Guide*. This means that you want all your functions to adopt the non-scrolling header style.

```
<RDML BUTTON="&EXIT">
 <a href="javascript:EmulateButton('HOME')">
<img src="/IMAGES/TB_HOME.GIF" alt="Home" border="0" width="55" ht
</a>
 </RDML>
<RDML BUTTON="&CANCEL">
 <a href="javascript:EmulateButton('MENU')">
<img src="/IMAGES/TB_MENU.GIF" alt="Menu" border="0" width="55" he
</a>
 </RDML>
<RDML BUTTON="&WEBEVENT">
 <a href="javascript:EmulateButton('OK')">
<img src="/IMAGES/TB_NEXT.GIF" alt="Next" border="0" width="55" heig
</a>
 </RDML>
```

The non-scrolling header style uses JavaScript functions. JavaScript support must be enabled in your browser.

# **Frameset Definition**

The non-scrolling style uses the frameset feature in HTML. Basically, it divides your browser into two frames. One frame holds the STDHEADER page while the other frame holds the body of your function.

The HTML used to define the frameset is stored in DEFAULT\_FRAMESET. This is the default frameset definition.

```
<RDML MERGE="&DTD_FRAMESET">
<!--
                        -->
<!--
     Default Frameset definition
                                 -->
     Last Modified: 9.1 2001-10-03 -->
<!--
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title><RDML MERGE="&FUNCTION"></title>
 <meta http-equiv="pragma" content="no-cache" />
</head>
<frameset rows="125, *" border="0" frameborder="no" framespacing="0" screeter</pre>
 <frame name="LHEADER" src="/<RDML MERGE="&CGI">/LANSAWE
PAGE+<RDML MERGE="&SESSION">+STDHEADER" />
 <frame name="LBODY" src="/<RDML MERGE="&CGI">/LANSAWEB?
PAGE+<RDML MERGE="&SESSION">+
<RDML MERGE="&PAGE">" />
 <noframes>
<body>You cannot view the documents with your current browser</body>
</noframes>
</frameset>
</html>
```

The <frameset> border, frameborder, framespacing and scrolling attributes are not part of the XHTML 1.0 frameset document type definition. They have not been removed to keep the appearance of existing applications.

You can modify this file or you can create a frameset definition for a particular LANSA process, by following the naming convention:

<process>\_FRAMESET

where <process> is the name of your LANSA process.

If a frameset page exists for the process, this page will be used instead of DEFAULT\_FRAMESET.

You can customize the height of each frame in this page. You can also choose to orientate the frames vertically, instead of horizontally. If you modify the DEFAULT\_FRAMESET or create a process specific page, do not modify the names assigned to the frames.

You will want to customize the height of the frames to optimize the usage of the browser's display area. The height of the header frame should be big enough to display the page. This will then provide maximum area to display the body of the function.

# Limitations of non-scrolling headers

If you intend to use non-scrolling header style in your Web function applications, you should be aware of some of the limitations of this style.

- If you use the Back button of your browser to display previous pages, this causes problems. The non-scrolling header style uses framesets and each frame is a self-contained page. The browser remembers the sequencing of the pages in its memory cache. It does not know the relationship of the frames.
- On a page request, LANSA for the Web will display the body of the function initially. It will then determine if the header needs to be refreshed. If it does, it sends a new header page to the header frame. This means that the last frame to be refreshed is the header frame. When you select the Back button of your browser, it will load the previous page of the header frame, without refreshing the body of the function. This may cause the header and the function to be out of synchronization.
- If you select the 'Refresh' or 'Reload' command in your browser, only the body of the function is loaded. The request to refresh does not reload the frameset definition.
- It is recommended that this header style is not used for WEBEVENT functions.

It is recommended that the non-scrolling header style be used only with Intranet or Extranet applications, where you can provide instructions to your users, and can control the type of browsers they use.

# **Combination Of Scrolling And Non Scrolling Header Styles**

LANSA for the Web allows you to have both scrolling and non-scrolling header styles in your Web function applications. You can customize your functions to have the different header styles.

However, if you want to have both scrolling and non-scrolling header styles, you will need to be very careful as a single STDHEADER component cannot be used for both scrolling and non scrolling header styles.

In the scrolling header style, the header component is embedded into the HTML page of the function.

In the non-scrolling header style, the header and the body of the functions are regarded as separate HTML pages.

Both header styles use JavaScript functions to emulate the selection of an action in the function. If JavaScript support is disabled in your browser, you will not be able to use the buttons from the standard headers.

If you intend to use a combination of scrolling and non-scrolling header styles in your application, you should spend time familiarizing yourself with the differences in the HTML used in the different header styles.

# **Override Header Style**

The setting Use Non Scrolling Header in the Presentation tab of the LANSA Administrator applies to all your functions. However, you can override this setting for specific LANSA processes. This allows you to have a combination of scrolling and non-scrolling header styles in your applications.

If you want to override the system setting for a particular LANSA process, you create an HTML page following the naming convention:

#### cess>\_FRAME

where <process> is the name of your LANSA process.

This style-override page only contains a single character, either a 'Y' or 'N'. A 'Y' character indicates that you want the non-scrolling header style enabled for the <process> process. A 'N' character indicates that you do no want the non-scrolling style enabled for the process.

Do not embed any comments at the start of this page. The first character in the page is used to determine the override.

LANSA for the Web will check if a style override page exists for the process. If it exists, it will be used to determine which header style to use in the function. It the page does not exist, LANSA for the Web will check the option set in the LANSA Administrator.

If the setting in your style override page is different from the global setting in the LANSA Administrator, you must ensure that there is a corresponding <process>\_STDHEADER component that is compatible with the style. Otherwise, your application may not work properly.

# **Adopted Header Style**

If your application contains functions that call other functions attached to different processes, the header style adopted for the application will be dependent on the first function called. This means that if the first function called is set up to have a non-scrolling header, then all the subsequent functions called from within the same LANSA job will adopt the non-scrolling header.

This means that the other functions called must also have the same header style. If the other functions have been set up to have different header styles, it would appear as though they have adopted the style of the first function. However, because of the conflicting HTML/JavaScript techniques used in the two different styles, your functions will not operate properly.

# Appendix B. LANSA for the Web XHTML

LANSA for the Web generates Web pages compatible with the XHTML 1.0 specification. The XHTML 1.0 standard and HTML 4.0 standards are almost identical. In this guide, the term HTML is used to describe LANSA for the Web generated HTML/XHTML pages. This section of the guide provides specific details about the XHTML generated by LANSA for the Web. For details about XHTML, review the following topics:

Introduction to XHTML 1.0

Converting HTML to XHTML 1.0

XHTML Document Type Definition (DTD)

Serving XHTML Pages as Pure XML Pages

# **Introduction to XHTML 1.0**

If you are new to XHTML, you should review: What is XHTML 1.0? Why use XHTML 1.0? Compatibility with HTML

# What is XHTML 1.0?

Following are some important facts about XHTML:

- XHTML is also referred to as Extensible-HTML.
- XHTML 1.0 is a reformulation of HTML 4.01 as an XML 1.0 application.
- The three XHTML 1.0 document type definitions (DTD) correspond to the ones defined by HTML 4.
- XHTML is designed to provide a path to extend HTML in the future in a way that is compatible with XML.
- XHTML is intended as a replacement for HTML.
- XHTML 1.0 is almost identical to HTML 4.01.
- XHTML 1.0 can be viewed with current generation user agents, including the major browsers if certain guidelines are followed. (Refer to Converting HTML to XHTML 1.0.)

## Why use XHTML 1.0?

There might not be an immediate business need forcing you to move to XHTML. However, it is recommended that you start using XHTML for new developments or modifications. This approach will make it easier to migrate to XHTML 1.0 or future versions of XHTML so that you can quickly take advantage of new business opportunities.

New user agents, like portable devices, will probably use newer versions of XHTML. Because XHTML conforms to XML syntax rules, the XHTML generated by LANSA for the Web can be processed by XML parsers.

The compatibility between XHTML 1.0 and HTML 4.0 allows you to continue to support all of your existing browser interfaces while allowing for future computing devices.

# **Compatibility with HTML**

Your existing HTML applications are compatible with the XHTML pages and markup language generated by the LANSA for the Web runtime.

Web pages can use a mix of both HTML and XHTML elements.

For details about the differences between HTML and XHTML, refer to Converting HTML to XHTML 1.0.

# **Converting HTML to XHTML 1.0**

XHTML 1.0 is a reformulation of HTML 4.01 as an XML 1.0 application. Hence, XHTML 1.0 is almost identical to HTML 4.01. In order to make your HTML compatible with XHTML, some very simple rules need to be followed.

Review the following:

XHTML Syntax Rules

XML Declaration

You may also need to refer to XHTML Document Type Definition (DTD).

# **XHTML Syntax Rules**

These guidelines will help you write Web pages that conform to the XHTML 1.0 transitional document type definition and that are compatible with current browsers.

### Documents must be well formed

All elements must be nested within the <html> root element. Sub elements must be in pairs and correctly nested within their parent element.

#### Root element namespace

The root element of the document must designate the XHTML namespace using the xmlns attribute. The namespace for XHTML is defined to be http://www.w3.org/1999/xhtml.

For example:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

#### Mandatory tags in all XHTML pages

These tags must be defined in all pages:

```
<html><head><title></head><body></body></html>
```

## Title element must be first element in header

This is wrong:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="pragma" content="no-cache" />
<title>Handle Banner Request</title>
</head>
```

This is correct:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Handle Banner Request</title>
<meta http-equiv="pragma" content="no-cache" />
</head>
```

#### All XHTML elements should be in lower case

For example: not <TABLE>.

## All attributes should also be in lower case

For example:

## All attributes must have values contained by single or double quotation marks

This is wrong:

This is correct:

## When possible, avoid multiple spaces or line breaks within an element's tags

Subject to line length constraints the W3C makes this recommendation because different user agents may treat white space and line breaks differently.

For example:

Do not write this:

```
<tablewidth="544"
border="0"
cellpadding="0"
cellspacing="0">
```

Write it like this:

#### All nonempty elements must have a closing tag

For example:

...

# *Empty elements must also be closed (with syntax for HTML compatibility)*

Leave a blank space before the self-closing '/' in empty element tags.

For example:

```
<img src="logo.gif" alt="Logo" border="0" width="189" height="23" /> <br />
```

### Elements must be nested correctly

```
For example:
```

<b>This is correct</b> <b>This is not correct</b>

## Attribute minimization is forbidden

This is wrong:

<dl compact> <input checked /> <input readonly /> <input disabled /> <option selected /> <frame noresize />

```
This is correct:
```

```
<dl compact="compact">
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

## The id Attribute replaces the Name Attribute

HTML 4.01 defines a name attribute for the elements a, applet, frame, iframe, img, and map. In XHTML the name attribute is no longer used and the attribute id is used instead.

To preserve compatibility with HTML browsers, use both attributes as in the

following example:

<img src="home.gif" alt="Return to home" border="0" id="home" name="hon

#### Replace ampersands in attribute values with character entity

This is wrong: <input type="button" value="Bob & Alice" /> This is correct:

<input type="button" value="Bob &amp; Alice" />

## <img /> tag must have the alt="" attribute

This is really an HTML 4.0 requirement. Because XHTML 1.0 is based on HTML 4.0, this is required.

This is wrong:

```
<img src="btn_next.gif" height="20" width="100" border="0" />
```

This is correct:

```
<img src="btn_next.gif" alt="Next" height="20" width="100" border="0" />
```

## Handle special characters in JavaScript

Special characters like < and & in scripts may be treated as start of markup by XML parsers. Also entities such as &amp; and &lt; will be recognized as entity references. To avoid this, the W3C specification suggests that you wrap the content of the script within a CDATA marked section so that it can be ignored by the XML parser.

Example according to the W3C:

```
<head>
<title>Script Test</title>
<script type="text/javascript" language="javascript">
<![CDATA[
function HomePage(){parent.location="/home/index.html";}
]]>
</script>
</head>
<body onload="HomePage()"></body>
```

However, when sent as HTML, the Java Script parsers in current browsers do not understand the CDATA keyword and may cause a syntax error. The solution is to comment out the CDATA keywords as a Java Script comment:

```
<head>
<title>Script Test</title>
<script type="text/javascript" language="javascript">
//<![CDATA[
function HomePage(){parent.location="/home/index.html";}
//]]>
</script>
</head>
<body onload="HomePage()"></body>
```

This should be used instead of the technique to comment out the scripts for browsers that do not support JavaScript using <!-- and //-->.

Summary: To comment out lines to the HTML parser <!-- --> is used. To write comments inside <script></script>, the // is used so it will be ignored by the JavaScript parser. To make the CDATA section invisible to the JavaScript parser, but visible to the XML parser, you use // <![CDATA[ and //]]>.

## Location of scripts

Scripts must be located either between the <header></header> or between the <body></body> tags.

#### **XML Declaration**

XML documents normally include an XML declaration in its first line. Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

According to the W3C recommendation, an XML declaration is not required in all XML documents. Unless you need your web pages to be handled as XML documents, do not include the XML declaration.

By default, LANSA for the Web does not include this declaration.

Warning: Some user agents will render the Web page as an XML tree instead of rendering the page as HTML content if they see the XML declaration. For more information, refer to Serving XHTML Pages as Pure XML Pages.

#### **XHTML Document Type Definition (DTD)**

The W3C standards for XHTML recommend that a DOCTYPE declaration be included in your documents. For more details, refer to Document Type Definition (DTD).

If your applications don't conform to the XHTML 1.0 specification and you want to avoid the insertion of the DOCTYPE declaration in your pages refer to How to Avoid the Insertion of the DOCTYPE Declaration.

## **Document Type Definition (DTD)**

According to the XHTML 1.0 W3C recommendation (http://www.w3.org/TR/xhtml1/) section 3.1.1, there must be a DOCTYPE declaration in the document prior to the root element.

The public identifier included in the DOCTYPE declaration must reference one of the three Document Type Definitions found in

http://www.w3.org/TR/xhtml1/#dtds using the respective Formal Public Identifier. The system identifier may be changed to reflect local system conventions.

<!DOCTYPE html

PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/DTD/xhtml1-strict.dtd">

<!DOCTYPE html

PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!DOCTYPE html

PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/DTD/xhtml1-frameset.dtd">

LANSA for the Web ships with default pages for each of these Document Type Definitions.

If you download the Document Type Definitions and install them on your web server, then you can change the DOCTYPE declarations so you don't need to refer to the w3.org web site.

Note: LANSA for the Web uses either the Transitional DTD or the Frameset DTD (for pages that contain frames). It does not use the Strict DTD, as strict XHTML requires that all style information be provided via style sheets and not embedded in the XHTML page.

Some old browsers may have problems avoiding the display of the DOCTYPE declaration. Current generation browsers correctly hide them from being displayed.

#### How to Avoid the Insertion of the DOCTYPE Declaration

If your applications do not conform to the XHTML 1.0 specification and you want to avoid the insertion of the DOCTYPE declaration in your pages, follow these steps:

#### Step 1. Replace the Transitional DOCTYPE declaration with a blank line

Using the LANSA for the Web Function Editor, modify the special page DTD\_TRANSITIONAL in your partition and replace it with a blank space (You cannot save an empty page).

#### Step 2. Replace the Frameset DOCTYPE declaration with a blank line

Using the LANSA for the Web Function Editor Editor, modify the special page DTD\_FRAMESET in your partition and replace it with a blank space (You cannot save an empty page).

## **Step 3. Do not include the RDML DTD merge tag in custom Spooled File Utility Pages**

If you have created your own custom versions of STDPRINT, STDREPORTLIST or STDREPORT pages, do not include the <RDML MERGE="&DTD\_TRANSITIONAL"> tag in those pages. The Spooled File Utility retrieves the special DOCTYPE pages from partition 'WEB' and language 'ENG' and not from your current partition.

#### Serving XHTML Pages as Pure XML Pages

The following information is provided if you are intending to serve your LANSA for the Web XHTML pages as pure XML pages instead of HTML/XHTML pages.

### Internet media type (MIME type) for XHTML

As of the publication of the XHTML 1.0 W3C recommendation, the general recommended MIME labeling for XML-based applications was not resolved. It makes no recommendation about MIME labeling for XHTML documents that do not need backward compatibility with HTML user agents.

Because you want to preserve compatibility with HTML user agents, continue using MIME type text/html.

Note: There are issues relating to the inclusion of the XML declaration in XHTML documents. In principle, user agents should rely on the MIME type to determine if a document should be handled as an HTML or XML document. However, some versions of current browsers "sniff" an incoming Web page and if they see the XML declaration, they handle the document as an XML document regardless of the HTTP MIME type sent by the Web server. To avoid this problem the shipped DTDs do not include the XML declaration.

If you want to serve your Web pages as pure XML documents, then you follow these steps:

#### Step 1. Add the XML declaration to the DOCTYPE pages

Using the LANSA for the Web Function Editor, add the XML declaration in the DOCTYPE pages DTD\_FRAMESET, DTD\_STRICT and DTD\_TRANSITIONAL. Use an encoding appropriate to your language's character set.

For example:

III LANSA HTML Editor - [DTD_TRANSITIONAL (ENG-0)]	
<u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u><u></u></u>	
D <b>≓⊟</b> 8885 5 % 68	
<pre></pre> <pre>&lt;</pre>	A
<pre><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0</pre>     </pre>	
	*
•	
Ready	Ln 1, Col 44 HTML L014550 DEM ENG DCXUSER

**Step 2. Change the LANSA for the Web server configuration** 

#### If you are using CGI-BIN:

1. Add the following row to table LWEB.DAT located in the LANSA shared library:

Column	Value
XLWPEN	'HTM'
XLWSEN	Blanks
XLWGRP	'MIMETYPE '
XLWLID	0
XLWMDF	Y

XLWDTA	text/xml
XLWSP1	Blanks

2. Uncomment the XML declaration in the PREAMBLE group. Add the encoding appropriate to your language's character set.

Column	Value
XLWPEN	'HTM'
XLWSEN	Blanks
XLWGRP	'PREAMBLE '
XLWLID	0
XLWMDF	Y
XLWDTA	xml version="1.0" encoding="ISO-8859-1"?
XLWSP1	Blanks

#### If you are using Java Servlet:

- 1. Add line MIMETYPE=text/xml to configuration file L4W3SERV.CFG.
- 2. Set the current directory to the home of the Java Servlet L4W3Servlet.jar.
- 3. Extract L4W3Resource.properties from jar file L4W3Servlet.jar (Select the properties file relevant to your locale). Use the Java command:

#### jar -xf L4W3Servlet.jar com/lansa/web/servlet/L4W3Resource.properties

4. Locate the properties file in the subdirectory tree com/lansa./web/servlet

under your current directory.

- 5. Change the value of property PR00 in the properties file, to uncomment the XML declaration.
- 6. Update the jar file. Use the Java command:

#### jar uf L4W3Servlet.jar com/lansa/web/servlet/L4W3Resource.properties

7. If your jar file is on an iSeries, you will need to recompile the jar file. Use the OS/400 command (replace path with the actual IFS path to the jar file):

#### CRTJVAPGM CLSF('/path/L4W3Servlet.jar') OPTIMIZE(40)