# **LANSA Composer Guide**

- What's New in this Version?
- Introducing LANSA Composer
- Define Integration Components
  - Activities
  - Activities by Group
  - Configurations
  - Trading Partners
  - Transformation Maps
- Processing Sequences
- Transaction Document Processing
- Using aXes Terminal Server with LANSA Composer
- Deploying Solutions for LANSA Composer
- Operations
- Administration and Housekeeping
- Develop Custom Activities for LANSA Composer
- LANSA Composer Tutorials
- Appendix A. Install LANSA Composer
  - LANSA Composer Server on IBM i
  - LANSA Composer Server on Windows
  - LANSA Composer Client on Windows
  - Uninstall LANSA Composer Client on Windows
- Appendix B. License LANSA Composer
- Appendix C. The Mapping Tool
- Appendix D. Commands to Invoke LANSA Composer
- Appendix E. Using LANSA Composer with LANSA Applications
- Appendix F. The LANSA Composer Request Server

• Appendix G. National Language and Multilingual Support in LANSA Composer

Edition Date April 29, 2014 This edition applies to Version 5.0 of LANSA Composer. © LANSA

## About this Guide

LANSA Composer is a design and execution platform for integrating business activities involving transport and transformation of data along with custom business processing. This guide describes the features and functions of LANSA Composer.

It is intended for use by business analysts who will use the application to design and implement solutions to business integration problems.

The text in this guide is available while using the LANSA Composer design environment by selecting the *Help* command from the *Help* menu on the LANSA Composer main window or by pressing F1.

The *LANSA Integrator Guide* is also provided with LANSA Composer and provides more in-depth technical information on the LANSA Integrator services that are used by many of the supplied LANSA Composer activities or that you may use in your own custom activities.

#### How to use this Guide

This guide is designed to get you started with the LANSA Composer software as quickly as possible. However, you will need to know some of the basic structures and windows of Composer before you begin. Please review Getting Started with LANSA Composer which includes a Quick Tour and Functional Components for an overview of LANSA Composer's components.

Sample configurations for supported transport protocols have been provided. You only need to refer to the description and specifications of the protocol that you will be using, and these are described in Configurations.

LANSA Composer is supplied with a number of Activities that will perform transport, file management and other common business integration functions for you. Refer to Activities for details.

Should you wish to develop some custom activities, you will find notes on how to do this in Develop Custom Activities for LANSA Composer.

#### What's New in LANSA Composer Version 5.0?

LANSA Composer version 5.0 introduces many new features and enhancements. The highlights are described in the following sections:

Licensing Installation New SQL Database Activities Other New and Enhanced Activities **Extended Duration Processing Sequences Cross References** Deployment Audit Trail **Events** Parameter Classes LANSA Composer Client User-interface Enhancements **Transformation Maps Transaction Document Support System Settings Database Housekeeping** Browsers for Server Files and Folders Other New and Enhanced Features

**Note:** Some features have been previously made available in hotfixes for LANSA Composer version 4.0.

## Licensing

**IMPORTANT NOTE:** 

Due to changes in the licensing implementation, all LANSA Composer customers upgrading from LANSA Composer version 4.0 must request and obtain new LANSA Composer server licenses before upgrading to LANSA Composer version 5.0. Prior to upgrading to Version 5.0, you should send your CPU details to LANSA Licensing for new Version 5.0 licenses. For more information refer to Appendix B: License LANSA Composer.

### Installation

# **LANSA Composer Client**

The LANSA Composer Windows client installer now offers the option of installing the LANSA Web Administrator program. (LANSA Web Administrator can be used, if necessary, to change the LANSA for the Web configuration on the LANSA Composer server.)

Altova MapForce version 2014 release 2 (2014R2) is provided on the media. It is strongly recommended that MapForce is upgraded to this level on all LANSA Composer clients.

# LANSA Composer Server on IBM i

The LANSA Composer installation on IBM i servers now supports installation into independent auxiliary storage pools (IASP), if required.

# LANSA Composer Server on Windows

The Windows server installation of LANSA Composer has been completely revised. It is now uses "standard" Windows installer/MSI technology, which will make future patches and upgrades simpler and more flexible. It also appears as a separate item in the Control Panel Programs and Features and can be uninstalled in the usual way.

Only a single copy of the LANSA Composer server may be installed on a Windows server.

LANSA Composer's database tables are now installed using a new and constant database schema/collection/owner (LC\_DTA), meaning that in future versions, the post-installation task to "cleanup" the previous collection will no longer be necessary (it is necessary for this version, however).

More of the post-installation tasks are now automated - in particular the installation of LANSA Composer's data directory and files. On the Windows server, the first client to connect will no longer need to complete the Server Initialization wizard.

# LANSA Runtime Version Information

LANSA Composer version 5.0 is built with LANSA version 13SP1 at EPC level 131100 and requires a run-time environment at least at that level. In addition it requires LANSA Integrator at EPC level 131300. This is of concern only to existing users who have installed LANSA Composer in an existing LANSA system. The upgrade of standard LANSA Composer installations (in their own

LANSA run-time system) will upgrade all the necessary components.

## New SQL Database Activities

This version of LANSA Composer introduces a new suite of twelve activities for performing SQL operations on a database identified by a LANSA Composer database configuration. The databases can be on any network addressable server. So, for example, LANSA Composer on an IBM i server could address an MS SQL Server database on a Windows server, or vice-versa..

Typically in LANSA Composer, most database activity is accomplished through the use of Transformation Maps. However, these new activities provide another option for performing limited database query and update operations on an SQL database. In addition, they provide a means of invoking SQL stored procedures in the target database.

The SQL database activities are not intended for high-throughput, high-volume database operations. Rather they provide a simple means to complete a business process integration solution that may not otherwise have been possible, with some simple, low-volume database access and/or maintenance.

The new SQL database activities comprise:

• Activities to establish or disconnect an SQL database connection:

SQL_CONNECT	Connect to database using SQL
SQL_DISCONNECT	Disconnect from database using SQL

#### • Activities to query the database:

SQL_QUERY	Query database using SQL
SQL_QUERYTOCSV	Query database using SQL to output CSV file

• Activities to perfom insert, update and delete operations in the database:

SQL\_UPDATE Update database using SQL

• Activities to execute an SQL stored procedure in the database:

SQL_CALL	Execute an SQL stored procedure
SQL_CALLQRYCSV	Query database using an SQL stored procedure to CSV
SQL_CALLQUERY	Query database using an SQL stored procedure

• Activities to implement transaction control relating to any database insert, update or delete operations you have performed:

SQL_COMMIT	Commit a database transaction using SQL
SQL_ROLLBACK	Rollback a database transaction using SQL

• Activities to set parameter values for an SQL operation:

SQL_PARAMS	Set parameter values for SQL operation
SQL_PARAMSCSV	Set parameter values for SQL operation from CSV

#### **Other New and Enhanced Activities**

In addition to the New SQL Database Activities, this version of LANSA Composer is supplied with a range of other new and revised Activities that deliver new and powerful capabilities:

New Activities Revised Activities

#### **New Activities**

The following new Activities are added in this version of LANSA Composer:

- The new BASE64\_ENCODE and BASE64\_DECODE activities will base64 encode or decode the contents of a specified file and write the encoded or decoded content to another file. Base64 encoding is commonly used when there is a need to encode binary data that needs to be stored and transferred over media or transports that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 encoding is commonly used in a number of applications including email via MIME, and storing complex data in XML.
- The new COUNTLIST activity counts the entries in a variable list used in a Processing Sequence. Previously it would have been necessary to iterate the list entries with a LOOP directive in order to count the entries.
- The new DTAQ\_CLEAR, DTAQ\_RECEIVE and DTAQ\_SEND activities provide support for sending and receiving data to and from IBM i data queues and to emulated data queues on Windows servers.
- The new FOR\_EACH\_OBJECT activity iterates for each object in an IBM i server library that matches a specified object name and/or object type. The activity is only supported on IBM i servers.
- The new HASH\_FILE activity generates a hash value of the contents of the specified file according to a known algorithm. The hash value is a fixed-length (according to the chosen algorithm), non-reversible representation of the contents of the file. Such hash values have many uses in information security, including as a means of detecting (accidental or intentional) changes to or corruption of the source data.
- The new JSM\_RECLAIM activity reclaims LANSA Integrator JSM resources by initiating garbage collection in the JSM's Java Virtual Machine instance. In some particular instances, a delay in garbage collection can lead to functional issues in subsequent processing when certain resources, such as

files, might remain locked awaiting garbage collection of Java objects that reference them. The JSM\_RECLAIM activity may help to avoid issues of this nature.

- The new QUERY\_CCSID activity returns the IBM i CCSID for a specified file. The activity is only supported on IBM i servers.
- The new WATCH\_DTAQ, WATCH\_MSGQ and WATCH\_DIRECTORY iterator activities provide a means of iteratively watching a data queue, message queue or a file system directory for new items (and/or changed or deleted items in the case of WATCH\_DIRECTORY) for further processing. The WATCH\_MSGQ activity is only supported on IBM i servers.
- The new WAIT\_FILESREADY activity is intended for use in processing sequences that identify and process newly-discovered files. For example, a processing sequence that uses the new WATCH\_DIRECTORY activity may need to use this activity. It provides file cache services designed to allow LANSA Composer solutions to control the processing of newly-discovered files to avoid contention issues with applications that may, for example, still be writing to a file when it is "discovered" by the LANSA Composer solution.
- The new XML\_QUERY activity permits a LANSA Composer solution to selectively interrogate values contained in an XML document using XML Path Language (XPath) expressions. The activity is intended for selective interrogation of a limited number of particular values from the XML document, perhaps to determine how to further process the XML document as a whole.
- The new XML\_VALIDATE activity validates an XML document file. At the minimum, the activity will verify that the XML document is well-formed. If the XML document contains or references a DTD or an XML schema and the DTD or schema can be accessed, then the document content will be validated against the DTD or schema.

#### **Revised Activities**

Several previously supplied Activities have been revised to add functionality and/or to support other new and revised functionality in this version of LANSA Composer:

• The COPY\_FILE, MOVE\_FILE, RENAME\_FILE and DELETE\_FILE activities have been revised to provide more diagnostic information in the event of failure on IBM i servers.

- The COPY\_FILE activity provides a new AUT parameter that specifies the method used to assign authority information to copied objects on IBM i servers.
- The DIRECTORY\_LIST activity provides new CONTENT and MAXDEPTH parameters that allow it to list either sub-directories *or* files contained in the specified directory, and, optionally, to include the contents of child or descendant directories.
- The FTP configurations and the supplied FTP activities, such as FTP\_INBOUND have been enhanced to provide support both for SSH compression and for SFTP public key authentication.
- The SMTP and POP3 configurations and the supplied email activities such as MAIL\_SEND and MAIL\_RECEIVE have been enhanced to provide support for explicit SSL/TLS.
- A new BACKIMAGEPATH and related parameters for the SPLF\_TO\_PDF activity permits a background image to be specified for inclusion in the generated PDF file. The SPLF\_TO\_PDF activity is only supported on IBM i servers.
- A new DOCCCSID parameter for the SPLF\_TO\_TEXT activity permits the solution to specify the IBM i CCSID with which the output text document is created. The SPLF\_TO\_TEXT activity is only supported on IBM i servers.
- The SYSTEM\_COMMAND activity has been revised to provide more diagnostic information including captured joblog messages in the event the command execution is unsuccessful on IBM i servers.

## **Extended Duration Processing Sequences**

A number of new features and enhancements are designed to further facilitate creation and management of extended duration Processing Sequences – for example, Processing Sequences that are intended to run indefinitely, perhaps monitoring a file system directory, data queue or message queue for new items to process. These changes include:

- New activities mentioned previously: WATCH\_DTAQ, WATCH\_MSGQ, WATCH\_DIRECTORY and WAIT\_FILESREADY.
- A new example Processing Sequence, EXAMPLE\_WATCH01, provides an example of a process using the new WATCH\_DIRECTORY and WAIT\_FILESREADY activities. It demonstrates some suggested design techniques for extended duration Processing Sequences.
- New documentation in the LANSA Composer Guide provides tips and suggestions for designing extended duration Processing Sequences in LANSA Composer. It can be found under the heading Considerations for Extended Duration Processing Sequences.
- Large processing sequence logs will usually load faster and more reliably. LANSA Composer reduces the initial loaded log size by "collapsing" detail associated with COMPLETED iterations for LOOP, WHILE/UNTIL directives and iterator activities, the premise being:

- that a long-running process will almost invariable involve iteration in some form, and;

- that the users interest when viewing the log will most often be on the latest or last iterations

Collapsed items will still be capable of being expanded, on demand.

- Associated with the previous item, the *Processing Sequence Log* window has been enhanced to have a better organized, consistent and simpler user interface.
- The *Processing Sequence Log* window more clearly identifies a processing sequence run that is active and now allows "controlled end" to be initiated for the run. Note the processing sequence must be designed to check and act on the value of the new \*SHUTDOWN built-in variable for such requests to be effective.
- When *Print* is selected in the *Processing Sequence Log* window, LANSA

Composer will print the log as displayed – that is with detail log messages shown or not and with items collapsed or expanded as they presently are in the *Processing Sequence Log* window.

## **Cross References**

A new Cross references command tab provides comprehensive cross reference information for most definition types in LANSA Composer, including:

- Activities
- Transformation Maps
- Processing sequences
- All types of transport and database Configurations

lame 🔶	Description	Details	Туре	Go to
Used by EXAMPLE_SQL01				
🚔 LOGLIST	Logs a variable list's values	ACTIVITY item, sequence 16	Activity	Befreel
SQL_CONNECT	Connect to database using SQL	ACTIVITY item, sequence 5	Activity	
SQL_DISCONNECT	Disconnect from database using SQL	ACTIVITY item, sequence 18	Activity	0.01
SQL_QUERY	Query database using SQL	ACTIVITY item, sequence 14	Activity	Print
SUBSTITUTE	Formats an input string with variable su	ACTIVITY item, sequence 12	Activity	
TXDOC_DATABASE	Transaction document database configu	PARAMETER item, sequence 1	Database configuration	
Uses EXAMPLE_SQL01				
DX MASTER EG	Shipped examples		Export list	

Using the cross reference information provided by LANSA Composer, you can quickly identify:

- Other definitions that the subject definition uses or refers to. For example, for a Processing Sequence, this would include Activities, Transformation Maps and configurations that are used in the Processing Sequence.
- Other definitions that use or refer to the subject definition. For example, for an FTP configuration, this would include Trading Partners and Processing Sequences that use the FTP configuration.

The cross reference information includes the much-requested references to configurations, providing the references are to a specific, fixed configuration. (References to configurations and other definitions using variables or built-in variables are resolved at run-time and cannot be cross-referenced.)

This new support provides a new level of impact analysis capability while maintaining your business process integration (BPI) solutions. In addition, it facilitates deployment of those solutions, for example, from a design environment to production.

(The *Cross references* tab applies to most definition types and it replaces the *Where Used* tab that was previously available for Activities and Transformation maps only.)

# Deployment

This version of LANSA Composer provides persistent and more detailed and informative logging for export and import tasks.

The logs for each export and import task are retained in the LANSA Composer database and can be retrieved and viewed at any time through the LANSA Composer client software by clicking the *History* tab.

Import Di History				
Run number	Start date/time	End date / time	Status	🕞 Refresh
00000000000038	2013-08-13 4:22:33 PM	2013-08-13 4:22:39 PM	🥝 OK	
000000000000037	2013-08-13 1:04:22 PM	2013-08-13 1:04:22 PM	🥝 OK	D. View
000000000000036	2013-08-13 1:01:01 PM	2013-08-13 1:01:02 PM	🥝 OK	
8 000000000000035	2013-08-13 12:57:15 PM	2013-08-13 12:57:17 PM	🥝 OK	Drint
00000000000034	2013-08-06 9:34:21 AM	2013-08-06 9:34:27 AM	🥝 OK	
00000000000033	2013-07-03 4:12:53 PM	2013-07-03 4:12:54 PM	🥝 OK	Manifest.
00000000000032	2013-07-02 8:06:14 AM	2013-07-02 8:06:17 AM	📀 ок	Second Second

From the *History* tab, you can also print an *Export Manifest* that lists all exported items along with their last-changed dates, times and users.



In addition, a number of other new and revised features enhance the reliability and useability of the existing LANSA Composer export and import capability. They include:

- Export and import of *System Properties* is now supported, including the option to preserve existing system property values on the importing system, if applicable
- Import provides the option to preserve the existing security credentials (user names and passwords) when importing configurations that already exist on the importing system.
- Import now updates the last-changed dates, times and user information for the imported definitions on the importing system to match the values with which the definitions were exported. This enables the information to be compared with a manifest or with the definitions on a source LANSA Composer system.

# Audit Trail

LANSA Composer now maintains an audit trail of significant, usually modifying events to most definitions, including Activities, Transformation Maps, Processing Sequences and all types of configurations. This enables you to trace when changes were made to definitions and by whom.

Events that are recorded include:

- Create, change, delete
- Opened for editing
- Saved (Processing Sequences)
- Prepared (Transformation Maps)
- Archived version restored, deleted or purged (Processing sequences and Transformation Maps)
- Exported, Imported

Information that is recorded for each event includes:

- The identity of item affected
- Timestamp
- Related log, where applicable (export, import)
- Job name, user and number
- Computer name

Note that the audit records do NOT record what actually changed in the item definition, only that the event occurred.

The existing Audit command tab for each of the definition types has been substantially revised to display the audit trail events for the selected item (instead of just the last changed date, time and user information that was shown before):

Processing sequence : A	Audit (EXAMPLE_SQL01-	Example of using t	he SQL_QUERY ac	tivity)	
📄 Details 📔 ≽ Run command	ls 📔 <u>]</u> Run history   🖶 Ve	rsion history 📔 📴 Cr	oss references 📔 🧷	Attachments 🛛 💟 Notes 🔤	O Audit
Internal identifier	3699851AC66E40A8	927862E1E824996A			🔄 Refresh
Last changed	2013-07-08 10:01:3	3 by user QOTHPRDC	WN		D <u>e</u> tails
Audit event	Timestamp *	Log	User name	Computer name	- Showlog
Exported	2013-08-22 12:35:57	00000000000075	QOTHPRDOWN	LANSAPC007	SHOW FOG
Opened for editing	2013-08-22 8:55:03 AM		QOTHPRDOWN	LANSAPC007	
Exported	2013-08-20 2:14:10 PM	00000000000074	QOTHPRDOWN	LANSAPC007	
Exported	2013-08-13 4:13:35 PM	00000000000072	QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:09:46		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:08:39		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:07:09		QOTHPRDOWN	LANSAPC007	
Exported	2013-07-08 10:07:26	00000000000066	QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-08 10:01:33		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 10:01:06		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-08 9:08:52 AM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 9:07:20 AM		OOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:40:52 PM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 3:38:35 PM		OOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:35:09 PM		OOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:33:53 PM		OOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 3:13:00 PM		OOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:12:36 PM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 2:53:15 PM		OOTHPRDOWN	LANSAPC007	
Changed (Details)	2013-07-05 2:52:32 PM		QOTHPRDOWN	LANSAPC007	
Created as copy of EXAMP	2013-07-05 2:52:17 PM		QOTHPRDOWN	LANSAPC007	

In addition, there is also a new *Audit trail* option appearing in the *Navigator*, under *Administration and Housekeeping*, that permits interrogation and viewing of the audit records across all definitions and types by various criteria:

Audit trail : Details						
By definition type By audit even	t By date By user By comp	uter				
Audit event	Timestamp 🔹	Name	Description	Log	User name	
🖃 🛗 Today						Details.
Archived version res	2013-08-22 1:16:00 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Archived version res	2013-08-22 1:15:54 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	Show Loc
Changed (Details)	2013-08-22 1:15:39 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Saved after editing	2013-08-22 1:15:25 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Opened for editing	2013-08-22 1:15:11 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Changed (Attachme	2013-08-22 1:15:04 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Changed (Notes)	2013-08-22 1:14:29 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Saved after editing	2013-08-22 1:14:06 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Opened for editing	2013-08-22 1:13:51 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Created as copy of	2013-08-22 1:13:47 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Export using export I	2013-08-22 12:36:10 PM	DX_MASTER_EG	Shipped examples	00000000000075	QOTHPRDOWN	
Exported	2013-08-22 12:36:10 PM	EXAMPLE_TP	Example trading partner d	00000000000075	QOTHPRDOWN	
Exported	2013-08-22 12:36:09 PM	EXAMPLE_SMS	Example SMS configuration	00000000000075	QOTHPRDOWN	+
۲.	III					

#### Events

LANSA Composer's Event Maintenance now permits you to add multiple event notifications for a single event. For example, one event may now both send an email and send a message to an IBM i message queue.

🤰 Event maintenance : D	etails (EXAMPLE_EVENT01-Example user-defined event)	
Details E Cross refere	ences   🖉 Attachments   🛄 Notes   🕟 Audit	
Name	EXAMPLE_EVENT01	<u>arr</u> <u>S</u> ave
Description	Example user-defined event	Refresh
Status	Active	🤣 Test
Click an event notification to sevent.	show its details below. Use the buttons to add, remove and arrange the notifications for this	<u>A</u> dd
Send an email		Remove
Send a message to an IBM	1 i message queue	Move up
Execute a specified function	a sequence	Move down

In addition, an event can now use any one or more of the following notification methods to provide the notification of the event:

- Send an email
- Send an SMS
- Send a message to an IBM i message queue (IBM i servers only)
- Run a specified processing sequence
- Execute a specified function

(Previously one and only one of email and SMS options could be chosen.) You can specify more than one instance of each notification method if you wish. Further enhancements to the event support include:

- You can now specify a list of email addresses for any of the TO, CC or BCC addresses for an email notification.
- A new Test button in the Event Maintenance window now permits you to quickly test the notifications for your event without having to contrive to run a failing processing sequence to do so.

# **Parameter Classes**

This version of LANSA Composer implements support for parameter classes for activity, transformation map and processing sequence parameters.

Each parameter can have a class associated with it that can specify what type of information is specified by the parameter. The available parameter classes include:

- File path
- Folder path
- Trading partner
- Transformation map
- Processing sequence
- Event handler
- Configurations of all types

If none of the above classes apply, then a default class of *General* is used.

For the most part, you need not be greatly concerned with the parameter classes, because:

- For supplied Activities, the parameter classes are shipped already set appropriately;
- For Transformation Maps, the parameter classes are inferred from the map definition during the *Prepare* step.

(However, you may wish to set parameter classes appropriately, where applicable, for your own Processing Sequences or for your own custom Activities.)

The important point to note is that this enhancement brings to LANSA Composer several important benefits:

- The use of parameter classes has facilitated more thorough and complete Cross References in LANSA Composer, most notably for configurations.
- The use of parameter classes provides in-place prompting for file and folder paths, configurations and other items, either when editing a Processing Sequence, or when you run an Activity or Processing Sequence through the LANSA Composer client software user-interface.

Conversely, you should also note:

- The parameter classes are (deliberately) not used for validation of parameter values. They do not restrict the values that can be entered for a parameter. (So you can still, for example, enter the name of an FTP configuration that you have yet to create.)
- The parameter classes do not impose any additional processing sequence run-time overhead.

To illustrate, the following partial screenshot from the *Processing Sequence Editor* shows the parameters for the FTP\_INBOUND activity in the detailer. You can see:

- The parameter class for each FTP\_INBOUND activity parameter is shown in the column on the right.
- A button for in-place parameter value prompting is visible in the currently active parameter, FTPCONFIG. When clicked, the button will allow the user to choose from a list of existing FTP inbound configurations.

		Activity Name: *	FTP_INBOUND - FTP Ge	t files from remote ho	st	9	
Det	ails	Parameter	s				
			Variable or value for parameter	Parameter name	Parameter description	Parameter class	
1	*	Inbound	'TC_FTPIN'	FTPCONFIG	FTP Configuration ID	FTP inbound configu	
2		Inbound	&REMOTE_DIR	REMOTEDIRECTOR'	Override Remote Directory to GET file fra	General	and the second
3		Inbound	&TC_DATA_OUT	LOCALDIRECTORY	Override Local Directory into which file pla	Folder path	III
4		Inbound		GETLIKE	Override 'GET selection' for filename masl	General	
5		Outbound		FILELIST	List of file retrieved - file name	General	
6		Outbound		FILEPATH	List of file retrieved - full path & name	File path	-

### LANSA Composer Client User-interface Enhancements

This version of LANSA Composer offers easier, faster and more flexible location and selection of the definitions in LANSA Composer that you wish to work with. At the same time, it can free up space in the LANSA Composer window by allowing you to hide the instance lists when not in use.

To accomplish this, the *Filters* and *Instance lists* used in previous versions have been replaced with unified Instance lists that integrate the filter/find features within them. This provides a more effective user-interface that is:

- More visually effective and appealing;
- Provides quicker access to search and locate items;
- Provides a greater range of means and search criteria to locate items;
- Integrates recently-visited, recently-modified and user-defined working lists;
- Implements much faster searches;
- Supports copy, delete and print on multiple selections;
- Capable of being "unpinned" and scrolled out of the way when not in use to provide more useable screen area for command handlers. In this unpinned state, the Instance list can slide back into view on demand, at the click of a button or with a single shortcut keystroke.

≈	Quick find p	rocessing sequences	P				
R	Recent Working lists All processing sequences						
AT A	Name Description		Description Last cha	anged Use	er 🔺		
	🖃 Recent	ly visited (local)					
4		EXAMPLE_AATEST1	Check LANSA Composer installation 2012-	09-17 XF4	1		
	63	EXAMPLE_AATEST2	Check LANSA Integrator functionality 2012-09	9-17 XF4	ŧ		
0	6	EXAMPLE_ACTPARM	Shows ways to provide activity parameter values 2012-09	9-17 XF4	ŧ		
	6	EXAMPLE_CSV01	Example of processing CSV 2012-09	9-17 XF4	ŧ		
	- i i i i i i i i i i i i i i i i i i i	EXAMPLE_LOG01	Logging Example 2012-0	9-17 XF4	ŧ		
	- <u>6</u>	EXAMPLE_LOOP01	Example of processing multiple documents. 2012-09	9-17 XF4	ŧ Ŧ		

For more information, refer to Locating and selecting items in the Instance Lists.

# **Transformation Maps**

There are several enhancements to the Transformation Map support in this version of LANSA Composer:

- Parameter Classes are automatically inferred for a Transformation Map during the *Prepare* step.
- LANSA Composer creates better human-readable Transformation Map parameter descriptions during the *Prepare* step.
- LANSA Composer is fully compatible with the latest (at time of release) MapForce versions, especially for "catalogued" map information.
- Transformation Map *Prepare* provides better diagnostic information, including date/time in the Prepare log, logging the server JSM instance path, and access to the IBM i server joblog when run against LANSA Composer server running on IBM i.

In addition, if you update to the latest version of the mapping tool (Altova MapForce version 2014 release 2), the following new capabilities will be available (relative to MapForce version 2012SP1):

- User-defined component names;
- Extended SQL-Where functionality: ORDER\_BY;
- MapForce supports logical files of the IBM i database and shows them as views;
- Support for SELECT statements with parameters
- Timeout settings for web service function calls and database execution
- Support for XML wildcards (xs:any and xs:anyAttribute)
- Support for comments and processing instructions in output XML
- Support for CDATA generation in XML files
- User-defined end-of-line settings for output files (\*)
- Greater control over the output of quote marks surrounding values in CSV files
- New function to calculate age based on a birthdate
- New functions for processing and generating sequences
- A new option for keeping useful connections after deleting a component
- Automatic highlighting of mandatory items in target components
- Improved mapping validation and editing help

• Support for Informix 11.7 databases and extended support for other databases;

(\*) Refer to Changed behaviour for text output files in MapForce 2013 Release 2 on IBM i servers for important information on the impact of this change on existing maps for LANSA Composer running on IBM i servers.

# **Transaction Document Support**

LANSA Composer's supplied Transaction Document Processing Framework has been extended to better cover the inbound and outbound transport operations:

- For inbound transport, the supplied "model" processing sequences (\*) have been extended to include a call to a new TXDOC\_RCV processing sequence for receiving transaction document files. That processing sequence, in turn, is set up to delegate to a processing sequence (modeled as TXDOC\_RCV\_TP) that is specific to the Trading Partner.
- For outbound transport, the supplied "model" processing sequences (\*) have been extended to include a call to a new TXDOC\_SEND processing sequence for sending transaction document files. That processing sequence, in turn, is set up to delegate to a processing sequence (modeled as TXDOC\_SEND\_TP) that is specific to the Trading Partner.
- Trading partner support has been extended to permit the trading partner specific *receive* and *send* processing sequences to be specified individually for each trading partner and new trading partner built-in variables are provided for accessing these values.

Receive processing sequence	TXDOC_RCV_TP	]]
Send processing sequence	TXDOC_SEND_TP	

Together, these changes make the Transaction Document Processing Framework, *as supplied*, a better fit for accommodating the varied transports associated with a range of trading partners, and should reduce the need to modify the framework solution for this purpose.

(\*) The changes are to the supplied processing sequences. Existing implementations will most usually have copied the supplied processing sequences and modified them to suit the specific implementation. In this case, the existing implementation may need to be revised to take advantage of this new support.

Other changes related to transaction document support include:

• The LANSA Composer Document Manager provides a *Resend* button for a completed outbound transaction document envelope. When clicked, the completed transaction document file will be sent again to the trading partner by means of executing a Processing Sequence identified by a new system setting. By default, the supplied TXDOC\_SEND processing sequence is

used, which in turn delegates to the processing sequence specified for the Trading Partner, if any.

- New \*txdoc.\* built-in variables provide access to transaction document envelope attributes in a Processing Sequence.
- A new application program interface (API) to replicate the functionality of TXDOC\_REGISTER and TXDOC\_STATUS customer application code.

### **System Settings**

Modifications to System Settings include:

- The *Server network path* system setting has been renamed to *Home path*.
- A new value introduced for *Home path relative to server* provides reliable path conversion where necessary between client-relative and server-relative forms.

Browse
Reset Value
Browse
Browse

In addition, LANSA Composer now provides an option to print the current system settings. You may be asked to do this in the context of a support call, for example.

System Settings now provides a new *System information* tab that provides information about the LANSA run-time environment in which LANSA Composer is running on your LANSA Composer server, including LANSA listener and JSM port numbers.

Listener port:	05592	
Web instance port:	05593	
LANSA Integrator (JSM) port:	05594	
LANSA Integrator (JSM) admin port:	05595	

### **Database Housekeeping**

Changes to Database Housekeeping facilitate the integration of this task into your scheduled operations, extend the housekeeping task with new sub-tasks, and reduce the time taken for the typical housekeeping task to complete:

- A command line interface is now provided for invoking the database housekeeping task on Windows servers. (Equivalent functionality was already available for IBM i servers via the COMPOSER command.)
- A new *Save* button in the user interface permits the housekeeping options to be saved without having to submit the task. These two changes facilitate the integration of this task into your scheduled operations.
- The database housekeeping task now optionally includes purging database records associated with the new Audit Trail.
- The database housekeeping task now optionally includes clearing the JSM trace and temporary files. These are temporary files that may have been created by the LANSA Integrator JSM (Java Service Manager) in the course of normal operations.
- LANSA Composer now provides persistent and more detailed and informative logging for database housekeeping tasks. The logs for each database housekeeping task are retained in the LANSA Composer database and can be retrieved and viewed at any time through the LANSA Composer client software by clicking the *History* tab.

Submit 3 History				
Run number	Start date/time	End date / time	Status	💮 Refresh
000000000000038	2013-08-06 8:00:48 AM	2013-08-06 8:02:50 AM	📀 ОК	
00000000000037	2013-08-06 7:59:24 AM	2013-08-06 7:59:49 AM	📀 ок	(). View
00000000000036	2013-08-06 7:56:50 AM	2013-08-06 7:57:25 AM	📀 ок	Sector Sector Sector
00000000000035	2013-08-06 7:54:13 AM	2013-08-06 7:54:58 AM	🕗 ОК	🔅 Print
00000000000034	2013-08-05 3:23:35 PM	2013-08-05 3:24:49 PM	📀 ок	

- The database reconcile and reorganize tasks have now been separated in the user interface such that each can be individually selected.
- The database reconcile task is unselected by default. In normal operations, this step should not be necessary and excluding it will significantly reduce the time to complete the database housekeeping job.

👕 Database housekeeping : Submit	
🗑 Submit 🛛 🔁 History	
Do you wish to purge the logs?	
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you want to keep entries with error status?	No
Do you wish to retain entries with warning status?	No
Do you wish to purge the version history?	
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you wish to purge the transaction document register?	
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you wish to purge the audit records?	
Last purged date and time	2013-08-06 8:00:49 AM
Number of days for which you wish to retain records	180
Do you wish to clear the JSM trace and temporary files?	
Last purged date and time	2013-08-06 8:02:48 AM
Do you wish to reorganize the database tables?	
Last reorganised date and time	2013-08-06 8:02:48 AM
Do you wish to reconcile the database ?	
Last reconciled date and time	2013-08-06 7:56:58 AM
NB: This can be a very time-consuming step that is not necessary in n	ormal circumstances

### **Browsers for Server Files and Folders**

The LANSA Composer client application now allows for much easier and more reliable selection or specification of server-side file and folder selections. This is achieved by implementing new browser windows that permit the user to browse the server file system directly.

	Name 🔶	Date modified	Туре	Size
A_Composer A_Composer A_Composer A_Composer A_Composer A_Composer A_Composer A_Composer BPI_WEBI Examples HTTPIN Imports Map TC_DATA TP Tutorial	Ack.txt EXAMPLE_SQL03.csv TUT_01_SKEL.txt TUTorder.dtd TUTorder.xml TUTORIAL.mdb	2010-07-05 1:40:49 AM 2013-08-20 3:00:30 PM 2010-07-05 1:40:58 AM 2010-07-05 1:41:07 AM 2010-07-05 1:37:37 AM 2009-01-08 5:47:29 AM	Text Document Microsoft Excel Text Document Document Type XML Document Microsoft Office	1 KE 1 KE 1 KE 1 KE 1 28 KE

The new browsers are used in a number of places in LANSA Composer, including:

- In the Server Initialization wizard and in System Settings;
- In various configurations to select server-side folder and file locations;
- To provide browsing support in connection with new Activity, Transformation Map and Processing Sequence Parameter Classes support where the *File path* or *Folder path* parameter classes are used.

### **Other New and Enhanced Features**

Amongst other revisions in this version of LANSA Composer are:

- The FTP configurations user interface has been reorganized for better clarity and simplicity.
- The FTP configurations and activities now support connection timeout and response timeout values, specified in milliseconds.
- HTTP Outbound configurations support a range of new values for the *Content Type* that applies to the file being sent by the HTTP\_POST activity. The new values include the special value Automatic that specifies that the activity should determine an appropriate content type according to the file extension of the file being sent. Automatic is now the default value for new HTTP configurations.
- The HTTP\_GET and HTTP\_POST activities will now retrieve the response content file, if requested, even when an error status is returned by the remote HTTP server. This enables the solution to receive and examine the response content to determine the details of the error, if applicable.
- A new *Test* button in Database Configurations provides a means to test the database configuration using the currently entered database configuration values (not necessarily saved values) in a similar fashion to that already provided for FTP and some other configuration types.
- The Processing Sequence Editor now supports drag-and-drop re-ordering of the processing sequence items.
- The Processing Sequence Editor previously had an option to show tree lines that more clearly show the relationship between processing sequence items. This option is now on by default, and the tree lines are also available in the Processing Sequence Log window and in printing support for both the Processing Sequence and the Processing Sequence Log.
- Improvements to processing sequence run-time performance have been achieved by eliminating, reducing or deferring some types of database operations and by other processing changes.
- Several groups of new built-in variables provide access to:
  - pending controlled end request (\*SHUTDOWN)
  - transaction document envelope attributes;
  - trading partner send and receive processing sequences;

- lists of trading partners by trading partner groups;

- timezone description and UTC offset values.
- LANSA Composer now provides a web service (SOAP) interface for running Processing Sequences.
- When connected to an IBM i server, the LANSA Composer client now provides direct access to the IBM i joblog for the associated LANSA server job from the client user interface in several places, including from the *Tools* menu, from the Transformation Maps *Prepare Log* window and from the *Processing Sequence Log* display (when shown in connection with processing sequence runs initiated through the LANSA Composer client).
- On IBM i servers, the COMPOSER REQUEST(ABOUT) command will now list the LANSA listener, HTTP listener and JSM port numbers in use.

# **1. Introducing LANSA Composer**

- 1.1 What Is LANSA Composer?
- **1.2 Functional Components**
- 1.3 Getting Started With LANSA Composer

# 1.1 What Is LANSA Composer?

LANSA Composer is a design and execution platform for integrating business activities involving transport and transformation of data along with custom business processing. It satisfies these three key requirements of a business process integration (BPI) solution:

## Transport

LANSA Composer provides ready-to-use Activities supporting widely-used transport protocols including:

- FTP Inbound (file transfer protocol)
- FTP Outbound (file transfer protocol)
- HTTP Inbound (hypertext transfer protocol)
- HTTP Outbound (hypertext transfer protocol)
- POP3 for inbound e-mail (post office protocol)
- SMTP for outbound e-mail (simple mail transfer protocol)
- Inbound and outbound via message brokering systems such as IBM MQ Series

To use these transport Activities with LANSA Composer, you simply need to provide configurations that specify the variable information required to connect or communicate with the other parties involved. There is no programming involved.

Using the supplied transport Activities, you can exchange business information such as orders and invoices with other parties, including external trading partners and internal business units, using agreed standards-based protocols as appropriate for each party.

# Transformation

LANSA Composer includes powerful visual mapping that allows you to define how to map data between disparate formats including:

- XML documents
- EDI documents
- Microsoft Excel 2010 (XLSX) files
- Text files (for example comma-separated values)
- Web service functions
- A wide range of databases, including IBM DB2 Universal Database for IBM i.

The maps created with the visual mapping tool can be directly integrated into your business processes along with transport and other Activities.

With the transformation capabilities you can transform business information between external and internal formats as it is received or before it is sent. In this way you can readily extract information from corporate databases to send as, say, XML conforming to a schema agreed with your trading partner. Similarly you can transform in-coming XML documents and write them directly to your corporate database for further processing by your existing applications.

#### **Process Orchestration**

LANSA Composer provides a simple but powerful process orchestration capability that allows you to combine transport, transformation and custom business processing with processing Directives that provide conditional, iterative and other capabilities. A complete business process consisting of a number of steps may be defined, executed and managed as a single processing unit of work, without programming.

#### Also see

- 1.1.1 Extensible And Customizable
- 1.1.2 Who Can Use LANSA Composer?
- 1.1.3 What Can I Do With LANSA Composer?
- 1.1.4 Transaction Document Processing with LANSA Composer
- 1.1.5 How Is LANSA Composer different to LANSA Integrator?

### 1.1.1 Extensible And Customizable

LANSA Composer can be extended and customized to accommodate organization-specific requirements. A simple interface is provided to enable you to "wrap" your specific business logic as Activities that may then be combined with transport and transformation provided by Composer into a single business process.

## 1.1.2 Who Can Use LANSA Composer?

LANSA Composer is designed to be used by business analysts to design and implement solutions to integration problems. It is not necessary to write program code to use Composer in solutions that involve standard transport and transformation activities. The implementation of the transport and transformation activities and the orchestration of them in business processes is all accomplished using highly-visual graphical interfaces.

LANSA Composer can, however, be extended to encompass custom business processing. Doing so will require Activity processes to be written to encapsulate the custom processing. Once these are defined in Composer as Activities, they are orchestrated in the same way as supplied transport activities and transformations.

So, LANSA Composer lends itself well to an environment in which the implementation of the custom business processing can be delegated to a services development group, while the combination and orchestration of the Activities is performed by business analysts.

# 1.1.3 What Can I Do With LANSA Composer?

LANSA Composer enables you to:

- Exchange business information and transactions in common and agreed formats with trading partners, internal business units and/or other business applications on the same or different computing platforms.
- Transform business information between XML, EDI, Microsoft Excel 2007 (XSLX), text and database formats, including the capability to invoke web services, whether they be publicly available, published by trading partners or within your organization for internal consumption
- Orchestrate the transport, transformation and other activities, pass variable data between them and apply conditional and structural Directives to create multi-step business processes that can be executed and managed as a single unit

These things can be entirely accomplished in a graphical, drag-and-drop environment by business analysts without having to write program code.

# A Simple Example

Consider the simple stereotypical scenario illustrated in the diagram:



In this scenario, your organization is required to periodically retrieve orders in an agreed XML format from your trading partner using file transfer protocol (FTP). More than one order document may be retrieved. Your application must process each order document to transform it from the XML format to your inhouse received orders database.

Such a scenario might be very easily implemented in LANSA Composer in three steps:

- 1.Create an FTP configuration that specifies the addressing details and security credentials necessary to communicate with the trading partner via FTP.
- 2.Create a Transformation Map that specifies how to map from the orders in XML format to the internal database.
- 3.Create a Processing Sequence that combines the supplied FTP\_INBOUND Activity with a loop that executes the Transformation Map for each received order document.

In LANSA Composer, the finished Processing Sequence for this simple scenario might look like this:

This Processing Sequence may then be invoked from your own applications or scheduled to run periodically through your job scheduler of choice as described in Run a Processing Sequence.

# **Extend The Example Scenario**

You can take and extend this scenario in many ways and still accomplish the result using LANSA Composer:

- Add custom business processing to validate and accept the incoming orders
- Generate acknowledgement documents in an agreed XML format and FTP them to the trading partner
- Implement similar processes for other types of business information such as inventory and price enquiries, purchasing, invoicing and more
- Use other transport protocols such as HTTP, IBM MQ Series or e-mail or implement different FTP configurations for different trading partners
- Design business processes that perform such processing repetitively for a range of trading partners

#### **1.1.4 Transaction Document Processing with LANSA Composer**

LANSA Composer's transport, transformation and process orchestration features provide an infinitely flexible and adaptable framework that can be used to solve almost any type of business process integration challenge.

However, many typical business process integration requirements follow a similar and familiar pattern involving the exchange of transaction documents, usually in standard forms such as XML or EDI, with known Trading Partners and transformations between those transaction document formats and an internal application database.

Because this pattern is so typical, LANSA Composer provides a ready-made framework solution that you can easily adapt and extend to encompass the exchanges and document types that are specific to your organization. As a part of this framework solution, LANSA Composer provides a transaction document registration database and the LANSA Composer Document Manager application that can be used to monitor and manage your inbound and outbound document flows.

LANSA Composer's transaction document processing framework and the LANSA Composer Document Manager can support virtually any type of transaction document exchange. However, particular support is provided for EDI X12 and EDIFACT transaction document formats.

## **1.1.5 How Is LANSA Composer different to LANSA Integrator?**

LANSA Integrator is a developer toolkit that enables integration of applicationto-application (A2A) and business-to-business (B2B) transactions through XML and Java services. Most uses of LANSA Integrator require a developer to write program code to access the integration services.

LANSA Composer uses LANSA Integrator services.

While performing very similar processes, LANSA Composer is designed for use by business analysts without the need to employ programming skills. A business analyst, with no programming skills, can compose business processes consisting of combinations of standard transport, transformations and other activities.

Many of the supplied Activities (including supplied transport activities) and the run-time execution of Transformation Maps make use of LANSA Integrator services but, using Composer, no program code is required to take advantage of them.

LANSA Integrator is designed to handle high-volume, high-throughput scenarios, so a custom application using LANSA Integrator services may offer the best performance characteristics for that type of application.

## **1.2 Functional Components**

The major functional components of LANSA Composer correspond to the key integration requirements of transport, transformation and orchestration. They are:

1.2.1 Activities

1.2.2 Transformation Maps

**1.2.3 Processing Sequences** 

Other components of LANSA Composer support and extend these to provide greater implementation flexibility:

1.2.4 Configurations

1.2.5 Trading Partners

## 1.2.1 Activities

Activities, along with Transformation Maps, implement specific business functions that are executed by LANSA Composer Processing Sequences. Examples of Activities include:

- Performing a file transfer via FTP (file transfer protocol)
- Zipping a file or folder
- Deleting a file

LANSA Composer is supplied with a set of Activities that provide transport, file management and zip/unzip capabilities. Refer to Supplied Activities for a complete list and description of the supplied Activities.

You can use LANSA Composer straight away with the supplied Activities. If required, you can extend LANSA Composer with custom Activities of your own.

## **1.2.2 Transformation Maps**

Transformation maps define how to transform or map data between disparate formats including XML, EDI, Microsoft Excel 2010 (XSLX), text files, web service functions and database tables.

Typically most of your structured corporate data is held in database tables, while information is exchanged with external parties in open formats such as XML documents. Commonly you will have the need to transform data between your internal database table representation and external exchange formats such as XML that you agree with your trading partners. Examples of transformation requirements include:

- Receiving incoming sales orders in agreed XML format and mapping to the received orders database
- Generating invoices in agreed XML format directly from your accounts receivable database tables
- Transferring information between internal applications (perhaps from different vendors) that are not directly integrated.

Transformation Maps created using LANSA Composer can be directly integrated into your business processes along with transport and other Activities.

## **1.2.3 Processing Sequences**

Processing sequences allow you to combine Activities and transformations with processing directives such as loops and conditions to complete a business process. Combining these elements with variable transport and database configurations and trading partner information permits great flexibility to accomplish complex multi-step processing without programming.

Processing sequences are defined visually in LANSA Composer using the *Processing Sequence Editor*. A typical Processing Sequence might be visualized in the editor like this:

		*** Beginning of processing sequence ***
÷	Activity	
	Name:	FTP_INBOUND - FTP Get files from remote host
÷	Activity	
	Name:	DEMO_BATCHNUM - Generate batch number
÷	Loop	
	List:	FILEPATH
	Loop variable:	THISFILE
Ē.	ᡖ Transform	
	Name:	DEM_ORDER_DB2 - Order.xml to DB2
÷	Activity	
	Name:	DEM_ORDER_RCV - Process received orders
	lf	
	Condition:	&ACKNOWLEDGEMENTS = 'YES'
Ē.	ᡖ Transform	
	Name:	DEM_DB2_ACK - DB2 to ack.xml
Ē.	land the second	
	Name:	FTP_OUTBOUND - FTP Put files on remote host
		*** End of processing sequence ***

Refer to Processing Sequences for further information.

# **1.2.4 Configurations**

Configurations encapsulate the variable information required for common transport Activities and for database connections. Typically configurations encapsulate information such as:

- Source and target identification and addressing and
- Security credentials

By encapsulating the implementation-specific information in configurations, LANSA Composer allows a single Activity and often a single Processing Sequence to be used with multiple variations of a business process (such as for multiple trading partners).

Specific configuration types support the supplied transport activities. They are:

- FTP Configurations
- HTTP Configurations
- SMTP Server Configurations
- SMTP Mail Details
- POP3 Configurations
- SMS Configurations
- Messaging Configurations

In addition, the special type of *Database Configuration* is provided to support encapsulating database details for databases used with Transformation Maps.

# **1.2.5 Trading Partners**

Trading partners are an optional feature of LANSA Composer that can be used to link variable information such as folders, configurations and Transformation Maps with specific trading partners.

Trading partners are particularly useful for:

- Organizing files into directories (typically for inbound and outbound transport),
- Sharing common activities and Processing Sequences with different source and target locations by using variable inputs associated with the trading partner
- Performing iterative processing for multiple trading partners

# **1.3 Getting Started With LANSA Composer**

#### **System Requirements**

Refer to LANSA Composer Requirements.

#### Installation

Installation instructions are in Install LANSA Composer.

As part of the install process, you need to set your required paths, defaults and codes as described in Administration and Housekeeping.

Once you have LANSA Composer installed and configured, you are ready to 1.3.1 Start LANSA Composer.

## 1.3.1 Start LANSA Composer

To start the LANSA Composer design environment, select the LANSA Composer shortcut that was installed on your start menu. LANSA Composer displays the *Connect to LANSA Composer server* window.

Connect to LANSA Composer s	erver		×
Connect		Connect	
<b>Composer</b>		Cancel	
	Work Dff		
Type the user name and password to co	onnect to your LANS	A Composer server.	
User	LICPGMLIB		
Password	*****		
Use Windows credentials			
Type or select the LANSA Composer ser	ver to which you wi	sh to connect.	*
Server connection name	COMPOSER01		-
Partition	LIC		
Click to specify connection attributes for	r your LANSA Compo	oser server.	*

Usually you simply need to type your *User* name and *Password*, type or select the server with which you want to work (if more than one server is available to you) and click the *Connect* button. In some cases, you may need to enter additional connection attributes the first time you connect to your LANSA Composer server. Refer to Connect to LANSA Composer Server for more information about this window.

If you are the first client to connect to your LANSA Composer server, you will see the LANSA Composer Server Initialization window. Refer to Step 1. LANSA Composer Server Initialisation for information if required.

Once you have successfully connected, the LANSA Composer *Start Here* window is displayed.



The *Getting Started* window provides quick access to some of the main areas of the LANSA Composer design environment, and, in particular, to those areas that you may wish to access as you are getting started with the software.

### **Connect to LANSA Composer Server**

The *Connect to LANSA Composer server* window lets you specify all the information necessary to make a connection to your LANSA Composer server. In many cases you need enter only your password and click *Connect*. However the first time you connect to your LANSA Composer server, you may need to enter additional information in order to complete the connection successfully.

The *Connect to LANSA Composer server* window is divided into three sections. The *LANSA Composer server* and *Server connection attributes* sections may be collapsed and hidden when not in use or expanded when required by clicking the up or down arrow buttons. LANSA Composer will remember the state of the window and the values used when you last connected (except the password) and present the window the same way the next time you start LANSA Composer.

Refer to the following headings for more information:

Type the User name and Password

Type or Select the LANSA Composer server

Specify Connection Attributes

Specify Windows Server Database Connection Attributes

Define Additional Servers Using the LANSA Communications Administrator

In the event of difficulties or errors completing the connection to your LANSA Composer server, please refer to the following:

Troubleshooting Connection Errors

#### Type the User name and Password

In the *User* and *Password* fields, you should type the user name and password assigned to you for connecting to your LANSA Composer server.

Alternatively, you can check the box labeled *Use Windows credentials* to use a Windows user profile and password when connecting to the LANSA Composer server. The server must have been configured for Single Sign On and the user enrolled first.

Type the user name and password to connect to your LANSA Composer server.		
User LICPGMLIB		
Password		
Use Windows credentials		

If you connect to more than one LANSA Composer server, you may need to use a different user name and/or password for each server.

When connecting to a new default installation of LANSA Composer server on IBM i, you can usually use the following credentials:

User **LICPGMLIB** (the default user name is the same as the *<programlibrary>* name in which LANSA Composer server was installed on IBM i)

Password Lansa

Note: If you wish to create and use your own user profiles to connect to your IBM i server, refer to Creating Further User Profiles for Use with LANSA Composer.

When connecting to a new default installation of LANSA Composer server on Windows, you can usually use the User Id (and its password) that you entered in the *User Id for Visual LANSA for Web Configuration* during the LANSA Composer Server forWindows installation. For example:

User PCXUSER

Password PCXUSER

**Note:** You may need to grant permissions to the database for the user in the database management system (for example, MS SQL Server) on the Windows server computer. This applies particularly if the database connection is using *trusted connections* and you are not separately specifying a database user and password in the *Server exceptional arguments* entry box.

#### Type or Select the LANSA Composer server

If the *Server connection name* and *Partition* fields are not shown, expand the dialog by clicking this line:



When you start LANSA Composer for the first time, the COMPOSER01 connection entry is pre-selected for you to connect to the LANSA Composer server system whose name and port number you specified during installation of the LANSA Composer client.

Type or select the LANSA Composer serv	ver to which you wish to connect.	\$
Server connection name	COMPOSER01	Ŧ
Partition	LIC	

If LANSA Composer server is installed in a partition other than the one shown, type the correct partition name. LANSA Composer initially shows the default partition name, LIC.

If you wish to connect to a different LANSA Composer server system, do one of the following:

- Select the desired *Server connection name* from the drop-down list (if you have connected this LANSA Composer client to the desired server before) and change the partition name if necessary
- Type the name of a different *Server connection name* that is defined in the *LANSA Communications Administrator* and change the partition name if necessary. If the connection is successful, the new *Server connection name* will be added to the drop-down list the next time you start LANSA Composer. (You can remove entries that you no longer use by clicking the remove button in the Connection attributes section of the window.)

Refer to Define Additional Servers Using the LANSA Communications Administrator for more information.

# **Specify Connection Attributes**

If the *Server connection attributes* fields are not shown, expand the dialog by clicking this line (you will need to expand the *LANSA Composer server* section first if it is not already expanded):

Click to specify connection attributes for your LANSA Composer server.

The *Server connection attributes* section lets you specify values that affect how LANSA Composer connects to the LANSA Composer server.

Specify connection attributes to connect	to your LANSA Composer server.
Server type	LANSA Composer server for IBM i
Client-server translation table	*JOB
Server-client translation table	*JOB
Selection block size	500
Selection limit	10000
Server exceptional arguments	
Open LANSA Commun	nications Administrator 🛛 💥

In most cases you will not need to change values of these fields. One notable exception, however, is when connecting to a LANSA Composer server for Windows, you may need to specify values in the *Server exceptional arguments* field that identify the server database/DSN (and credentials if applicable) used by the LANSA Composer server. These values depend on choices made when LANSA Composer server is installed.

- Refer to LANSA documentation topic *X\_Run Parameter Summary* for complete information concerning all the possible keywords and arguments values you can use in the *Server exceptional arguments* field.
- Refer to Specify Windows Server Database Connection Attributes below for more information concerning the database connection keywords and arguments you might need to specify to successfully connect to a LANSA Composer server on Windows
- If in doubt, refer to your system administrator for the correct values to use for your system.

#### **Specify Windows Server Database Connection Attributes**

NOTE: This section concerns connections to a LANSA Composer server on Windows. If you are connecting to LANSA Composer server on an IBM i server you do not need to read this section.

The following example shows default and typical values used in the *Server exceptional arguments* field when connecting to a LANSA Composer Windows server that uses an MS SQL Server database:

Specify connection attributes to connect	to your LANSA Composer server.	*
Server type	LANSA Composer server for Windows	-
Selection block size	500	
Selection limit	10000	
Server exceptional arguments	DBID=*SERVER DBII=*SERVER	
Open LANSA Commun	nications Administrator	×

NOTE: In the above example no database user credentials were supplied because the MS SQL Server database was configured to use Windows authentication. Depending on your configuration, you may need to specify DBUS= and PSWD= keywords and values in the Server exceptional arguments string to provide database user authentication credentials.

In many cases, the default values shown will be sufficient to successfully connect to your LANSA Composer server database. However, in exceptional cases, you may need to alter the *Server exceptional arguments* database values to suit your particular server configuration.

The server exceptional arguments that most usually are used to connect to the LANSA Composer server database are listed in the table. You should refer to LANSA documentation topics *X\_Run Parameter Summary* and *DBID*, *DBUT*, *DBII and DBIT Parameters* for more complete information.

Keyword Description		Examples
DBID=	Specifies the database/DSN id of the LANSA Composer database on the	DBID=*SERVER
	Windows server.	or
	Usually you can use the special value *SERVER to specify that the database/DSN name is specified the <i>x_lansa.pro</i> file on the server, which can be found in the .\ <i>X_WIN95\X_LANSA</i> folder of the Windows server application installation.	DBID=LCDB01
DBII=	Specifies the database/DSN id of the	DBII=*SERVER

	LANSA Composer database containing the LANSA internal/repository database tables. Normally, this should be the same as the DBID= value.	or DBII=LCDB01
	(NOTE: LANSA Composer does use certain LANSA internal tables for messages, locking and other purposes. Certain LANSA Composer operations will not succeed if the DBII= value is not specified correctly.)	
DBUT= DBIT=	These keywords may be provided to specify the type of the database specified by the DBID= and DBII= keywords, respectively.	DBUT=SQLANYWHERE DBIT= SQLANYWHERE
	The default value is MSSQLS (for a Microsoft SQL Server database). If your LANSA Composer Windows server database is of another type you may specify:	
	ORACLE SQLANYWHERE	
DBUS= PSWD=	These keywords may be provided to specify the user name and password for the database login.	DBUS=DBA PSWD=SQL
	If your MS SQL Server database is configured to use <i>integrated login</i> , you usually do not need to specify these keywords (but instead, you must ensure that the user is configured in the database to enable the integrated login to succeed).	

In other cases, the DBUS= and/or

PSWD= keywords may already be specified in the *x\_lansa.pro* file on the server, which can be found in the .\*X\_WIN95*\*X\_LANSA* folder of the Windows server application installation. If this is the case, you usually do not need to specify them again here.

What follows is additional information concerning the way in which the server exceptional arguments affecting the database connection are used in LANSA Composer:

• When the LANSA Composer client starts (and before it connects), the DBID= value is initially \*NONE and the other values have LANSA default values (usually). This is appropriate for the execution of the client software, but it usually means that the DBID= keyword must be explicitly specified (at least as DBID=\*SERVER) for the connection to the LANSA Composer server.

Note that the DBID= keyword value may be affected by the contents of the LANSA Composer's *dxstart.cfg* file, which can be found in the  $X_WIN95X_LANSA$  because folder of the client application installation.

(An exception to the preceding occurs for the default installation of the *offline* client used as part of the LANSA Composer Windows server installation - in this case, the *dxstart.cfg* file is usually pre-configured with database values appropriate for direct access by the *offline* client.)

• For the server connection, LANSA Composer pre-fills the Server exceptional arguments with the following defaults:

#### DBID=\*SERVER DBII=\*SERVER

In most cases this is appropriate and will cause the LANSA Composer server connection to use values specified in the *x\_lansa.pro* file on the server, which can be found in the .\*X\_WIN95*\*X\_LANSA* folder of the Windows server application installation.

• Depending on your specific configuration, the DBII= keyword and argument

value can be important for the successful operation of LANSA Composer and should usually be specified in addition to DBID= (they should usually have the same value).

LANSA Composer does use certain LANSA internal tables for messages, locking and other purposes. Certain LANSA Composer operations will not succeed if the DBII= value is not specified correctly.

Depending on your specific configuration and choices, the contents of the *x\_lansa.pro* file on the server may be crucial to the success of the LANSA Composer Windows server connection. If you use the special value \**SERVER* for any of the keywords, or, in some cases, if you omit the keywords, the applicable values will be retrieved from the *x\_lansa.pro* file on the server. The *x\_lansa.pro* file can be found in the .\*X\_WIN95*\*X\_LANSA* folder of the Windows server application installation.

The following table shows example values for Server exceptional arguments when connecting to the specified databases. You will need to alter these to suit your database or DSN names, system configuration and environment. You should refer to LANSA documentation of the X\_RUN parameters for more information.

MS SQL Server	DBID=*SERVER DBII=*SERVER
ORACLE	DBID=LX_LANSA DBUT=ODBCORACLE DBUS=PCXUSER PSWD=PCXUSER
Sybase SQL Anywhere	DBID=LX_LANSA DBUT=SQLANYWHERE DBUS=PCXUSER PSWD=PCXUSER

#### **Define Additional Servers Using the LANSA Communications Administrator**

If you have more than one installation of LANSA Composer server, you can create new *Server connection name* entries in the LANSA Communications Administrator for each server. Once each is defined in the LANSA Communications Administrator, you can choose the server to connect to when starting LANSA Composer as described in Type or Select the LANSA Composer server.

To define additional servers using the LANSA Communications Administrator,

follow these steps:

- 1. Fully expand the *Connect to LANSA Composer server* window, if not already expanded.
- 2. Click the Open LANSA Communications Administrator button.
- 3. In the LANSA Communications Administrator window, click New.

Host Route Information			
Use generated LU Name for TCP hostna Partner LU Name (Name of the Host to which You Want to Co Fully Qualified Name of the Host (Address):	nme/port add	dress pair. COMPOSER02 SERVER02	
Communications Method:		Socket	•
C Options			1
Leave these fields blank to use the defau	lts.		ок
Connection Identifier:	4545		
Communications Module:			<u>C</u> ancel
Maxl_Field (Packet Size):			
Enable TCP_NODELAY TCP/IP so	, icket option		Help
Enable IPv6			

4. Complete the fields in the *Host Route Information* window as follows:

Partner LU Name	Type a symbolic name that you wish to use as the new <i>Server connection name</i> . For example, you might use COMPOSER02 as the name for your second LANSA Composer server.
Fully Qualified Name of the Host	Type the DNS name or the IP address of the server system containing the required LANSA Composer server installation. For example, SERVER02 or 192.168.192.2.
Connection Identifier	Type the port number used for the LANSA listener in the required LANSA Composer server installation. For example, 4545.

5. Click OK to save the new *Host Route Information* details and then close the *LANSA Communications Administrator* window.

- 6. In the *Connect to LANSA Composer server* window, type the new *Server connection name* using the name you specified in the *Partner LU Name* above. Then type the partition name and any further server connection attributes necessary to connect to that LANSA Composer server.
- 7. Type the *User* and *Password* that apply to the new *Server connection name* and click *Connect*.

If the connection is successful, the new *Server connection name* will be available in the drop-down list the next time you start LANSA Composer.

### **Troubleshooting Connection Errors**

Common errors that may occur during the connection to the LANSA Composer server are discussed under the following headings:

Could Not Logon to LANSA Composer server Could Not Connect to LANSA Composer server LANSA Composer Database Access Not Available Possible LANSA Composer Server Configuration Problem

## **Could Not Logon to LANSA Composer server**

When you attempt to connect to your LANSA Composer server, you may see a message like this:



Usually this error simply means that the user name and password you typed at the *Connect to LANSA Composer server* window are not correct for the server system.

You should check the following:

- Did you type the user name correctly? Is the user name correct for the server system? If necessary check with your system administrator.
- Did you type the password correctly? On some systems, the user name and/or password are case-sensitive. If this is the case, check that you typed the details in the correct case and that the Caps Lock is not on.

On some systems, especially on IBM i servers, the user may become disabled for various reasons - including repeated failed logon attempts. Check on the server system that this has not occurred for the user with which you are trying to connect.

#### **Could Not Connect to LANSA Composer server**

When you attempt to connect to your LANSA Composer server, you may see a message like this:

~	Could not connect to LANSA Composer server COMPOSER01.
A	
	Common causes of this error include:
	<ul> <li>the communications link to the server is not available;</li> </ul>
	<ul> <li>the listener is not started on the server system;</li> </ul>
	- the server name is not correct;
	the client system (in the LANSA Communications Administrator).
	the energy seen (in the Barton common editors / animistrator)
	(More information: communications error code 20 received.)

If you have not connected using the specified details before, then you should first check:

- that the *Server connection name* specified matches the *Partner LU name* of an entry in the *LANSA Communications Administrator*.
- that the host route information specified in the *LANSA Communications Administrator* is correct, especially the *Fully qualified name of the host* (address) and the *Connection identifier* (port number).

Refer to Define Additional Servers Using the LANSA Communications Administrator for more information.

If you are still having difficulties, check that:

- the communications link is operational (is your computer connected to the network?)
- the LANSA listener for the LANSA Composer server system is active on the server.

For information on starting the LANSA listener for LANSA Composer on an IBM i server, refer to:

Subsystems and Jobs

For information on starting the LANSA listener for LANSA Composer on a Windows server, refer to:

#### Services used by the LANSA Composer Windows Server

#### LANSA Composer Database Access Not Available

When you attempt to connect to your LANSA Composer server, you may see a message like this:



This message means that the LANSA Composer client successfully established the communications link to the LANSA Composer server but not the connection to the LANSA Composer server database. You must correct the condition that prevented the database connection in order to complete the connection to the LANSA Composer server.

Some possible causes of this error may be:

- The credentials used to connect to the database are not correct. If you are using an MS SQL Server database with *integrated login* configured and you have not provided database credentials by means of the DBUS= and PSWD= keywords of the Server exceptional arguments, then this may mean that the user has not been configured in the MS SQL Server database to allow the integrated login.
- The database connection has not been correctly specified in either the *Server exceptional arguments* or the *x\_lansa.pro* file on the server.
- The database is not presently accessible or the server-side service or process that provides the database access has not been started.

For more information refer to the following topics:

Specify Connection Attributes

Specify Windows Server Database Connection Attributes

#### **Possible LANSA Composer Server Configuration Problem**

When you attempt to connect to your LANSA Composer server, you may see a message like this:



This message means that the LANSA Composer client successfully established the communications link to the LANSA Composer server and was able to access the LANSA Composer server database, but it could not exchange BLOB data with the server or perform database operations involving BLOB data. You must correct the condition that prevented the BLOB operation in order to complete the connection to the LANSA Composer server.

(BLOBs are "binary large objects". LANSA Composer uses BLOB support to save and access file attachments for LANSA Composer definitions of all types. In particular, this support is also used to catalogue Transformation Map definition (\*.mfd) and implementation (\*.jar) files when various operations are performed on Transformation Maps.)

The most usual cause of this error is that the designated server temporary file location used for sending and receiving the BLOB files is incorrect or the user on the server does not have the necessary permissions. For example, in a default configuration, the LANSA Composer server may be attempting to use C:\Windows\Temp as the temporary file location for BLOB files, but if the user on the server does not have the necessary permissions to create and/or read folders and files in this location then the BLOB operations will fail. If this is the issue, there are two suggested remedies:

• Grant the necessary permissions to the designated server temporary file

location used for sending and receiving the BLOB files; OR

• Designate an alternate server temporary file location (with the necessary permissions) to be used for sending and receiving the BLOB files. One way to do this is to add a line specifying the 'LPTH=' parameter to the x\_lansa.pro file in the LANSA Composer Server installation.

The x\_lansa.pro file may be found in the x\_win95x\_lansa directory of the LANSA Composer Server installation – for example,

C:\Program Files\LANSA\LANSA Composer Server\x\_win95\x\_lansa. To modify this file, open it in a plain text editor, such as notepad.exe.

In order to designate C:\Temp as the alternate server temporary file location, you could add a line like this to the x\_lansa.pro file, save your changes and then stop and restart the LANSA Composer listener service.

LPTH=C:\Temp

### 1.3.2 Composer Quick Tour

LANSA Composer presents a consistent interface in which many of the main user interface elements are common across most of the major functions of the program. The following screen picture shows the LANSA Composer window when *Processing Sequences* are selected in the *Navigator*.



Like most Windows applications, LANSA Composer offers menus and tool bar buttons near the top of the window for quick access to functions of the program or to perform common editing operations. At the bottom, there is a status bar that displays messages and status information, including the connected state. In the main body of the window, you will usually find:

Navigator

Instance Lists

**Command Handlers** 

Refer also to the following headings:

Locating and Selecting Items in the Instance Lists

#### Navigator

The *Navigator* provides access to the major functional areas of LANSA Composer. You can simply click on an item in the tree to switch to working in that area of LANSA Composer.



Note: Some items in the Navigator may not be displayed due to access restrictions. Refer to Configure User Access for more information.

### **Instance Lists**

The *Instance Lists* show items of the type selected in the *Navigator* (for example, Processing Sequences) and allow you to select an item to work with.

*	Quick find p	processing sequences	P			
R	Recent Working lists All processing sequences					
A A	Name		Description	Last changed	User	
8	Recently visited (local)					
		EXAMPLE_AATEST1	Check LANSA Composer installation	2012-09-17	XF4	
-	63	EXAMPLE_AATEST2	Check LANSA Integrator functionality	2012-09-17	XF4	
Ø	6	EXAMPLE_ACTPARM	Shows ways to provide activity parameter values	2012-09-17	XF4	
	6	EXAMPLE_CSV01	Example of processing CSV	2012-09-17	XF4	
	6	EXAMPLE_LOG01	Logging Example	2012-09-17	XF4	
	· 63	EXAMPLE_LOOP01	Example of processing multiple documents.	2012-09-17	XF4	-

For more information about using the Instance Lists, refer to the following heading:

Locating and Selecting Items in the Instance Lists

## **Command Handlers**

The command handler area is where you work with an item after selecting it in one of the instance lists. Typically you will work with the *Details* for the item but you can choose different commands for the item using the context menus, toolbar buttons or the tabs across the top of the command handler area. When the focus is in the command handler area, you can also use *Alt+PageUp* and *Alt+PageDown* keyboard shortcuts to quickly switch between the available tabs.

🐐 Processing sequence : Details	: (EXAMPLE_AATEST1-Check LANSA Composer installation)	E
Details 📄 Run commands 🛛 🛅	🕽 Run history   🖶 Version history   🏪 Cross references   🖉 Attachments	s   🛄 Notes   📀 Audit
Name	EXAMPLE_AATEST1	Save
Description	Check LANSA Composer installation	🔄 Refresh
Restartable	Yes	▶ R <u>u</u> n
Status	Active	
Event identifier		
Click Edit to edit y	Eġt	

Note that some types of items display a lot of information. In some cases (depending on screen and window dimensions) you may have to scroll the command handler area to see all the information.
### Locating and Selecting Items in the Instance Lists

The *Instance Lists* and the associated *Find* support provide a great deal of flexibility to find items that you wish to work with. This section provides an overview of the features in the following sections:

Organization of the Instance Lists Selecting a Current Item to Work With Pinning or Unpinning the Instance Lists Finding Items Using the Quick Find Box Finding items Using the Find window Using Working Lists

# **Organization of the Instance Lists**

Several lists are available (depending on the definition type selected in the *Navigator*) that are organized in various ways, providing a very flexible means of locating and selecting the items with which you wish to work. For example:

• The *Recent* tab shows items (of the current type) of recent interest in two groups:

- the *Recently visited* group shows items that you have recently displayed in the LANSA Composer client at this computer. Whenever you make an item the "current" item, the item will move to the top of this group.

- the *Recently modified* group shows items that have been recently created or modified on the LANSA Composer server (that is, at any client computer).

- The *Working lists* tab shows one or more working lists of items that you can build yourself by adding items that appear on other tabs.
- When you use the *Find* or *Quick Find* features, the matching items will be shown in the *Find results* tab. (The *Find results* tab is not shown until you perform a find.)
- The *All* <*items*> tab shows all the available items in alphabetical groups.

≫	Quick find p	rocessing sequences		Ģ			
R	Recent W	/orking lists All process	sing sequences				
T A	Name		Description		Last changed	User	
8	🖃 Recent	ly visited (local)					
		EXAMPLE_AATEST1	Check LANSA	Composer installation	2012-09-17	XF4	15
-	63	EXAMPLE_AATEST2	Check LANSA In	tegrator functionality	2012-09-17	XF4	
Ø	6	EXAMPLE_ACTPARM	Shows ways to p	provide activity parameter values	2012-09-17	XF4	
	6	EXAMPLE_CSV01	Example of proc	essing CSV	2012-09-17	XF4	
	- <u>6</u>	EXAMPLE_LOG01	Logging Example		2012-09-17	XF4	
	6	EXAMPLE_LOOP01	Example of proc	essing multiple documents.	2012-09-17	XF4	-

Depending on the type of definition you are working with, additional tabs may be available. For example, when working with Activities, an *Activities by group* tab is shown.

While the input focus is on the instance lists, you can press *Alt+PageUp* and *Alt+PageDown* keyboard shortcuts to quickly switch between the available tabs. Of course, you can also click on the tabs with your mouse.

# Selecting a Current Item to Work With

The item that is currently selected (the "current" item) is identified in the instance lists in bold and with a different (right-arrow) icon.

≈	Quick find p	rocessing sequences	P				
8	Recent	/orking lists All process	ng sequences				
(a)	Name		Description		Last changed	User	
æ	🖃 Recent	y visited (local)					
	• •	EXAMPLE_AATEST1	Check LANSA Comp	oser installation	2012-09-17	XF4	
-	6	EXAMPLE_AATEST2	Check LANSA Integrate	or functionality	2012-09-17	XF4	
Ø	1 🚳	EXAMPLE_ACTPARM	Shows ways to provide	activity parameter values	2012-09-17	XF4	
	1 🚳	EXAMPLE_CSV01	Example of processing	CSV	2012-09-17	XF4	
	1 🚳	EXAMPLE_LOG01	Logging Example		2012-09-17	XF4	
	6	EXAMPLE_LOOP01	Example of processing	multiple documents.	2012-09-17	XF4	-

Note that the "current" item may not always be present in the particular instance list you are working with. In this case it is not specifically identified in that list, but you can still identify it by looking at the bar above the command handler area.



To work with an item that you have located in the Instance Lists, you need to make it the "current" item.

By default you do this by simply selecting it (click it with the mouse or use the up and down arrow keys until the desired item is select).

If you prefer, you can configure LANSA Composer such that an item becomes the "current" item only after you select it and press *Enter*, or after you double-click the item. To do this, right-click in the instance lists, select *Options* from the popup menu, and then select *Set current item immediately* to select or deselect this option.

n	2012-09-17 9:41:0	QOTHPRDOW
	2012-09-17 9:41:02	QOTHPRDOWN
er values	2012-09-17 9:41:02	QOTHPRDOWN
	2012-09-17 9:41:02	QOTHPRDOWN
issing	2013-08-13 8:22:41	XF4HUGH
F	2012-09-17 9:41:03	QOTHPRDOWN
IS.	2012-09-17 9:41:03	OOTHPRDOWN
Compo	eer installation)	
	er values ssing	2012-09-17 9:41:02 r values 2012-09-17 9:41:02 2012-09-17 9:41:02 ssing 2013-08-13 8:22:41 2012-09-17 9:41:03 s. 2012-09-17 9:41:03

Either way, when you make a new item the current item, information relating to the item will be shown in the command handler area.

<table-of-contents> Processing sequence : Details</table-of-contents>	(EXAMPLE_AATEST1-Check LANSA Composer installation)	
Details 📔 Run commands 🛛 🗍	Run history   🖶 Version history   🏪 Cross references   🖉 Attachments	🛄 Notes 🛛 🕢 Audit 🛛
Name	EXAMPLE_AATEST 1	<u>Save</u>
Description	Check LANSA Composer installation	Refresh
Restartable	Yes	Run
Status	Active	
Event identifier		
Click Edit to edit y	our processing sequence.	Edit

# **Pinning or Unpinning the Instance Lists**

When they are first shown, the *Instance Lists* will be pinned open. This means they permanently occupy space above the command handlers. In this state, you can resize the instance lists by clicking and dragging in the space between the instance lists and the command handler area.

The *pinned open* state is indicated by this button in the tool buttons to the left of the instance lists:

₿

If you wish, you can unpin the instance lists to free up more space on your screen for information shown in the command handler area. To do this, click the pin button (or use the corresponding context menu item). The instance lists will be removed from the screen, leaving only this visible:

In this unpinned state, you can temporarily display the instance lists at any time by using the Ctrl+F11 keyboard shortcut or by clicking this button:

When you display the instance lists in an unpinned state, they will be automatically hidden again as soon as the input focus moves away from the instance lists, for example, if you click in the *Navigator* or in the command handler areas or if you switch to another application.

To pin the instance lists open, click the pin button again.

# Finding Items Using the Quick Find Box

If you know part or all of the name or description of an item you wish to work with, you can find it very quickly using the *Quick Find* box.

To access the *Quick Find* box, use the Ctrl+Shift+F keyboard shortcut or click in the *Quick Find* box:

```
V Quick find processing sequences
```

Type some text that you expect to find in the names or descriptions of the items you wish to locate and press Enter. Any matching items will be displayed on the Find results tab.

Name         Description         Last chan.           Quick find results - 'receive inbound' 3 processing sequences found         2012-11-1           TXDOC_RCV         Receive inbound transaction document(s)         2012-11-1           TXDOC_RCV_DEMO         Receive inbound transaction document(s) for demo         2012-11-1	Worki	rking lists	Find results	All processing sequences		
Quick find results - 'receive inbound' 3 processing sequences found     TXDOC_RCV Receive inbound transaction document(s) 2012-11-1     TXDOC_RCV_DEMO				Description	Last chan	User
TXDOC_RCV     Receive inbound transaction document(s)     2012-11-1     TXDOC_RCV     DEMO     Receive inbound transaction document(s) for demo     2012-11-1	ck find r	d results - '	receive inbound	d' 3 processing sequences found		
TXDOC BCV DEMO Receive inhound transaction document(s) for demo 2012-11-1	🚰 TXE	XDOC_RC	V	Receive inbound transaction document(s)	2012-11-13	XF4.
	🚳 TXE	XDOC_RC	V_DEMO	Receive inbound transaction document(s) for demo	2012-11-13	XF4.
TXDOC_RCV_TP Receive inbound transaction document(s) for TP 2012-11-1	🚳 TXE	XDOC_RC	V_TP	Receive inbound transaction document(s) for TP	2012-11-13	XF4.

If only one item matched, then it will be automatically selected as the "current" item and the details will be shown in the command handler area.

# Finding items Using the Find window

You can use the *Find* window to find items using a more extensive set of search criteria than just the name and description.

To access the *Find* window, use the Ctrl + F keyboard shortcut or click this button::

Ø

The *Find* window provides a text search similar to Quick Find, but also provides a range of other search criteria in groups that you can expand and collapse according to your requirements and interest.

	Specify criteria to find proce	essing sequences in one or	Eind
$\mathcal{O}$ ;	nore of the groups below. I processing sequences mate	Click Find to show the ching the criteria.	🔽 Close <u>o</u> n find
a .	Chaose how to show the find results:		Reset
Lnoose no			
Replace	nrevious find results		LIDSE
ype text to	find in processing sequen	ices	4 L <u></u>
ype text to	o find in processing sequen	ices	J (
ype text to     In <u>n</u> am	e find in processing sequen	nces n description	
ype text to v In <u>n</u> am lick to spec	e find in processing sequen	ndescription	
ype text to In <u>nam</u> lick to spec	o find in processing sequen e I I I ify items to find in process ify last-changed details to	nces In <u>d</u> escription ing sequences find in processing sequences	

The groups of search criteria vary a little depending on the type of definition you are working with. For example, when working with Activities, a group is provided to allow you to find activities that belong to a specified activity group.

If you specify more than one search criterion, they must *all* be satisfied in order for an item to be matched.

When you have specified your search criteria, click the Find button. Any matching items will be displayed on the Find results tab.

R	Recent   Working lists   Find	results All proc	essing sequences		
÷	Name	Description	Last d	an	User
	Find results	3 processing s	equences found a document file solit 2012-0	9-17	XE4
	TXDOC_IEDI090	EDI transactio	n document acknowledgement 2012-1	1-12	XF4
6	TXDOC_INBOUN	Process inbour	nd transaction documents 2012-1	1-13	XF4

### **Using Working Lists**

You can build your own static working lists of items of interest to you. These working lists are stored only on your computer.

To add a selected item (or items) to a working list, click this button in the tool buttons to the left of the instance lists:

LANSA Composer lets you choose an existing working list to add the item(s) to, or you can specify the name for a new working list.

Note: The working list names are common across all LANSA Composer's definition types. So if you create a new working list name for Activities, then the same working list name will be added to the *Working lists* tab for Trading Partners and other definition types.

To display your working lists and the items contained in them, click the *Working lists* tab:

Recent Working	lists All processing sequences	
Name	Description	Last chan User
🗄 My first workin	ng list	
My other work	ing list	
🚳 EXAMP	PLE_CSV01 Example of processing CSV	2012-09-17 XF4
🚳 EXAMP	PLE_LOG01 Logging Example	2012-09-17 XF4
Working list 1		

For further options to maintain your working lists, including options to remove items and delete lists, use the context (right-click) menu at the *Working lists* tab.

# Working with Definition Items

Using LANSA Composer you can create, maintain, and delete any number of definitions of the following types according to your application's needs:

- All the major definition types comprising Activities, Trading Partners, Transformation Maps and Processing Sequences;
- All configuration types (for example, FTP configurations and HTTP configuration)
- Export Lists
- Document types and standards
- Event definitions

To work with any of these definition types, simply click the entry in the *Navigator* for the desired type.

The *Instance Lists* for that type will be shown. Refer to Locating and Selecting Items in the Instance Lists.

Each definition type is described in more detail elsewhere in this guide, but many of the ways in which you work with your definitions are common to all definition types. These are briefly described in this section under the following headings:

Create a New Definition Copy or Delete Definitions Review or Change a Definition Print Definitions and Lists of Definitions Review Cross references Maintain Attachments Define or Review Notes Review Audit Details

# **Create a New Definition**

Use the *Ctrl*+*N* shortcut or click the *New* button on the toolbar to create a new definition. Then complete the details for the new definition as described elsewhere in this guide for the definition type. Remember to click *Save* before switching elsewhere in LANSA Composer to save the details of your new definition.

## **Copy or Delete Definitions**

You can copy or delete one or more definitions at a time. Select one or more items in the *Instance Lists* and then click the *Copy* or *Delete* buttons or use the provided menu items or shortcuts.

LANSA Composer will show a list of the selected items and wait for you to confirm the copy or delete request. For a copy request, you must type a new name for each item to be copied before proceeding.



(If no item is selected in the current instance list when you request *Copy* or *Delete*, LANSA Composer will prompt you to copy or delete the "current" item – that is the item for which details are presently shown in the command handler area).

NOTE: when you copy items that contain security credentials (user names and/or passwords), LANSA Composer will copy the user names but will not copy the passwords. You will need to revise each copied definition to enter the correct security credentials as required.

# **Review or Change a Definition**

To work with an item that you have located in the Instance Lists, simply select it (depending on instance list options chosen, you may need to press *Enter*, or double-click the item). Information relating to the item will be shown in the command handler area.

Often the *Details* command handler tab will be presented first, showing the basic identifying information and definition for the item, but other command tabs may be shown. The command handler tabs that are common to all definition types are briefly described under the following headings:

Maintain Attachments

### Define or Review Notes Review Audit Details

In addition, other command handler tabs may be available depending on the type of definition with which you are working. Refer to the description for the definition type elsewhere in this guide for specific information.

While the input focus is in the command handler area, you can press *Alt+PageUp* and *Alt+PageDown* keyboard shortcuts to quickly switch between the command handler tabs available for the definition type you are working with. Of course, you can also click on the tabs with your mouse.

# **Print Definitions and Lists of Definitions**

You can print the definitions of most definition types in LANSA Composer, including:

- Activities
- Transformation Maps
- Processing sequences
- All types of transport and database Configurations

You have several printing options:

- 1. To print the definition of an item with which you are currently working, click the *Print* button in the toolbar or use the *Ctrl+P* keyboard shortcut.
- 2. To print the definitions of several items at once, select the items in the *Instance Lists*, click the *Print* button on the *Instance List* and then select *Print all selected definitions* from the popup menu.
- 3. To print just a list of items shown on the current *Instance Lists* tab, click the *Print* button on the *Instance List* and then select *Print this list* from the popup menu.

Depending on the current print options (which can be changed using the *Print Options* command from the *File* menu), the output is shown in the *Presentation Viewer* window (as shown below), or in print preview mode, printed immediately or shown in your web browser.

When the *Presentation Viewer* window is open, you can use the menu commands and toolbar buttons to change the format and/or level of detail shown, and to save, print or preview the output or to open the output in your web browser.



### **Review Cross references**

LANSA Composer provides comprehensive cross reference information for most definition types in LANSA Composer, including:

• Activities

- Transformation Maps
- Processing sequences
- All types of transport and database Configurations

Using the cross reference information provided by LANSA Composer, you can quickly identify:

- Other definitions that the subject definition uses or refers to. For example, for a Processing Sequence, this would include Activities, Transformation Maps and configurations that are used in the Processing Sequence.
- Other definitions that use or refer to the subject definition. For example, for an FTP configuration, this would include Trading Partners and Processing Sequences that use the FTP configuration.

NOTE: only static references in a Processing Sequence are crossreferenced. If a Processing Sequence refers to an Activity, Transformation Map or a configuration using a variable or a built-in variable, then such references are resolved at run-time and cannot be cross referenced.

To review cross reference information for a definition such as a Processing Sequence:

- 1. Select the definition type in the *Navigator*.
- 2. Use the *Instance Lists* to locate and select the item whose cross references you wish to review.
- 3. Click the *Cross references* tab to display the cross references window.

lame 🔶	Description	Details	Туре	Go to
Used by EXAMPLE_SQL01				
🍰 LOGLIST	Logs a variable list's values	ACTIVITY item, sequence 16	Activity	Befred
SQL_CONNECT	Connect to database using SQL	ACTIVITY item, sequence 5	Activity	- Henes
SQL_DISCONNECT	Disconnect from database using SQL	ACTIVITY item, sequence 18	Activity	A D.
SQL_QUERY	Query database using SQL	ACTIVITY item, sequence 14	Activity	Print
SUBSTITUTE	Formats an input string with variable su	ACTIVITY item, sequence 12	Activity	
TXDOC_DATABASE	Transaction document database configu	PARAMETER item, sequence 1	Database configuration	
Uses EXAMPLE_SQL01				
DX MASTER EG	Shipped examples		Export list	

You can print or refresh the cross reference information by clicking the appropriate buttons.

You can switch directly to one of the listed items by double-clicking the item or by selecting it and pressing Enter or clicking the *Go to* button.

# **Maintain Attachments**

You can attach files to most definition types in LANSA Composer, including:

- Activities
- Transformation Maps
- Processing sequences
- All types of transport and database Configurations

You can attach any kind of file that you can access through the file system of your computer. LANSA Composer will hold the file attachment(s) in its database, permanently associated with the specific definition to which you attached it. When you export and import your LANSA Composer definitions, the attachments will be included (subject to export and import options).

You might use this feature to attach such things as:

- A copy of an XML Schema used in a Transformation Map;
- A description of operating procedures relating to a Processing Sequence;
- Contact information and/or pictures or copies of trading agreements with a Trading Partner.

To review, add, change or delete attachments associated with a definition such as a Processing Sequence:

- 1. Select the definition type in the *Navigator*.
- 2. Use the *Instance Lists* to locate and select the item whose attachments you wish to work with.
- 3. Click the *Attachments* tab to display the attachments window.



Remember to *Save* your changes when you are finished and before switching elsewhere in LANSA Composer.

### **Define or Review Notes**

You can create notes for most definition types in LANSA Composer, including:

- Activities
- Transformation Maps
- Processing sequences
- All types of transport and database Configurations

Notes can be used to expand upon the short description of an item to further describe its intended purpose and use. For example you can describe the function of a custom Activity and the purpose and requirements for its parameters.

The notes can later be reviewed to remind you or others of when and how to use the item. Notes for Activities, Processing Sequences, Transformation Maps and Configurations can be displayed when editing Processing Sequences in the *Processing sequence editor*.

Notes are supplied for most of the Activities, Configurations and other definitions that are shipped with LANSA Composer.

To review or change the notes:

- 1. Select the definition type in the *Navigator*.
- 2. Use the *Instance Lists* to locate and select the item for which you wish to access the notes.
- 3. Click the *Notes* tab or the *Notes* button on the toolbar to display the notes window.

The notes window displays the existing notes, if any. You can edit the notes as required.

```
Notes 🛛 🐼 Audit
📄 Details 🛛 🔁 Parameters 🗌
                                    Groups
                                                     Run history Q. Where used
                                                                                                                                       2 Save
                                                                                                                              ~
FTP INBOUND
 _____
This activity retrieves a list of files from a remote host using FTP.
It connects to the remote FTP host and retrieves files from a directory
on the remote host. The files are placed in a local directory.
Details of the remote host and directories is taken from an FTP
                                                                                                                                         Refresh
configuration.
formats of the files retrieved is output. The list is output in two
formats - one containing just the file
names the other containing the full local path and file name.
An email Event notification named FTPINFAILED is available in this
activity. If this event is active and a failure occurs in this activity,
then an email will be sent.
INPUT Parameters:
FTPCONFIG : Required
This parameter should contain the name of an FTP configuration on file 
DXFC. This configuration can be created and maintained using the FTP
Configuration option in the Transport Configurations Business Object.
If this parameter is not provided or is not found, this activity will 
return an error and processing will be abandoned.
```

Remember to *Save* your changes when you are finished and before switching elsewhere in LANSA Composer.

**Note:** You can display but you cannot change the notes for Activities supplied with LANSA Composer.

# **Review Audit Details**

Most definition types in LANSA Composer have audit information that records by whom and when the item was changed and when other significant operations were performed, such as importing and exporting the item.

To review the audit information for an item:

- 1. Select the definition type in the Navigator
- 2. Use the *Instance Lists* to locate and select an item.

3. Select the *Audit* tab or the *Audit* button on the toolbar to display the audit information window.

nternal identifier	3699851AC66E40A	8927862E1E824996A			🔄 Refrest
ast changed	2013-07-08 10:01:3	33 by user QOTHPRDO	WN		D <u>e</u> tails
Audit event	Timestamp *	Log	User name	Computer name	Show Loo
Exported	2013-08-22 12:35:57	00000000000075	QOTHPRDOWN	LANSAPC007	SHOW FOG!
Opened for editing	2013-08-22 8:55:03 AM		QOTHPRDOWN	LANSAPC007	
Exported	2013-08-20 2:14:10 PM	00000000000074	QOTHPRDOWN	LANSAPC007	
Exported	2013-08-13 4:13:35 PM	00000000000072	QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:09:46		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:08:39		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 11:07:09		QOTHPRDOWN	LANSAPC007	
Exported	2013-07-08 10:07:26	00000000000066	QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-08 10:01:33		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 10:01:06		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-08 9:08:52 AM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-08 9:07:20 AM		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:40:52 PM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 3:38:35 PM		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:35:09 PM		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:33:53 PM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 3:13:00 PM		QOTHPRDOWN	LANSAPC007	
Saved after editing	2013-07-05 3:12:36 PM		QOTHPRDOWN	LANSAPC007	
Opened for editing	2013-07-05 2:53:15 PM		QOTHPRDOWN	LANSAPC007	
Changed (Details)	2013-07-05 2:52:32 PM		QOTHPRDOWN	LANSAPC007	
Created as copy of EXAMP	2013-07-05 2:52:17 PM		OOTHPRDOWN	LANSAPC007	

The audit information window displays the following information:

#### **Internal Identifier**

You assign a human-readable identifier (name) to definition items such as Activities and Configurations. But internally, LANSA Composer assigns a unique internal identifier to help to ensure that components can be transferred between systems irrespective of whether the user-assigned identifiers exist on the target system. For most purposes, you do not need to know or use this internal identifier.

#### Audit events

Shows a list of the changes and significant events for this item.

You can also interrogate and view the audit trail across all definition types and items using the *Audit Trail* option under *Administration and Housekeeping* in the *Navigator*.

# **2. Define Integration Components**

- 2.1 Activities
- 2.2 Activities by Group
- 2.3 Configurations
- 2.4 Trading Partners
- 2.5 Transformation Maps

# 2.1 Activities

Activities encapsulate individual units of work or business processes that can be combined with Transformations in a Processing Sequence. Activities, along with Transformations, implement the business functions that are executed by LANSA Composer Processing Sequences. Examples of Activities include:

- Performing a file transfer via FTP (file transfer protocol)
- Zipping a file or folder
- Deleting a file

Two key aspects of the definition of an Activity support their implementation and use along with other Activities and Transformations to orchestrate a complete business process:

• Activity Processor This is a LANSA re-useable part that implements the function of the Activity. LANSA Composer invokes the processor as required in order to execute the Activity. The Activity Processor is provided for all supplied Activities. If you extend LANSA Composer with your own Activities, you will write their Activity Processors according to your requirements.

#### • Activity parameters

The inbound and outbound parameters for an Activity provide the variable information necessary for an Activity to complete its work. More than that though, they provide the communication between different Activities and Transformations that are orchestrated together in a single Processing Sequence. Thus, the output (outbound parameters) from one Activity might provide the input (inbound parameters) for the next.

LANSA Composer is supplied with a set of ready-to-use Activities that provide transport, file management and zip/unzip capabilities. Refer to 2.2.17 All Supplied Activities for a complete list and description of the supplied Activities or to 2.2 Activities by Group for a quick reference list of the Activities you may need for a particular task.

You can use LANSA Composer straight away with the supplied Activities. If required, you can extend LANSA Composer with custom Activities of your own. Refer to Develop Custom Activities for LANSA Composer for information on developing your own custom Activities.

#### **Iterator Activities**

Iterator Activities are a special form of Activity that can be used to perform iterative processing in a Processing Sequence.

It is possible to perform iterative processing using standard Processing Sequence directives such as LOOP, WHILE and UNTIL. Iterative processing performed in this way is controlled by:

- The extent of a variable list (LOOP directive)
- A conditioning expression (WHILE or UNTIL directives)

Iterator Activities extend this capability to enable iterative processing controlled by any application-defined data, rule or condition. When you add an Iterator Activity to a Processing Sequence, it becomes a "block" item (similar to a LOOP for example) under which you can nest further Processing Sequence directives, Activities or Transformation Maps that are to be executed for each iteration of the Iterator Activity.

Some of the Activities supplied with LANSA Composer are Iterator Activities. They include:

FOR\_EACH\_CSVROW Iterate for each row in a CSV file

FOR\_EACH\_INDEX Iterate for each index in a range

FOR\_EACH\_TXDOCO Iterate for each pending outbound transaction doc

You can extend LANSA Composer with your own Iterator Activities. Such Iterator Activities might be driven by data in your own applications. For example, you might define and implement Iterator Activities that iterate for:

- Each division in your organization
- Each of your products in a specified product group
- Each product delivery scheduled on a given date

If you define your own Iterator Activities. you must implement the Activity Processor in a particular way such that it supports the iterative behavior. Refer to Develop a Custom Activity Processor for more information.

If there is any problem with any supplied activity, refer to Restore Supplied Definitions for information on how to restore them.

# 2.1.1 Work With Activities

Using LANSA Composer you can create, maintain, and delete Activity definitions according to your application's needs.

To work with Activities, expand *Definitions* in the *Navigator* and click *Activities*. To find out how to locate and select Activities to work with, refer to:

Locating and selecting items in the Instance Lists

For information on common tasks associated with Activities (such as creating, copying, deleting and printing definitions) refer to:

Working with definition items

For information on tasks associated specifically with Activities, refer to the following headings:

Review or change 2.1.3 Activity Parameters for your activity

Review or change 2.1.4 Activity Groups to which your activity is assigned Run an Activity

Display the Run History of an Activity

**Note:** You can review the definitions of Activities that are supplied with LANSA Composer, but you cannot change most of the information. You cannot delete supplied Activities.

**Note:** If you create your own custom Activities, you must also supply an Activity Processor that implements the custom processing. Refer to Develop a Custom Activity Processor for information on developing your own custom Activity Processor.

#### **Run an Activity**

You can run an existing Activity directly, without having to first create a Processing Sequence containing the Activity.

To do so, select the required item in the Activities list. Details of the selected Activity will be displayed. Click the *Run* button to run the activity. LANSA Composer will display a window like the *Run Processing Sequence* window in which you can enter the input parameter values for the activity.

Refer to Run a Processing Sequence from LANSA Composer.for further information on using the *Run Processing Sequence* window.

Running an Activity in this way is very much like running a Processing Sequence containing just that Activity. LANSA Composer will automatically log the input and output parameter values in the Processing Sequence log (subject to the logging level currently in effect).

#### Display the Run History of an Activity

To display available Run History for an Activity, select the required item in the Activities list and select the *Run History* tab. A list of processing sequence runs (including direct runs of the Activity) that used the selected Activity will be displayed.

Select an item and click the *View* button to display the Processing Sequence log or the *Print* button to print the Processing Sequence log for the run.

If a prior Processing Sequence run has ended in error, you may be able to use the *Restart* button to restart it from the point of failure. Refer to Restart a Processing Sequence Run for more information.

**Note:** The completeness of the Run History is subject to the logging level that was in force for each run. Only at more detailed levels of logging is the log information sufficient to identify each and every Activity run. In addition, Processing Sequence run history can be purged using the Database Housekeeping function. The run history for Processing Sequence runs that have been purged is no longer available.

# 2.1.2 Activity Details

The Details tab identifies the Activity and contains basic information.

Details 🛛 😤 Parameters	🕼 Groups   🔃 Run history   🔍 Where used	🛄 Notes 🛛 🐼 Audit
ID	DEMO_BATCHNUM	Save
Description	Generate batch number	Refresh
Status	Active	► Run
Supplied activity	No	
Activity processor	DXACTDEMO	]
Keep active	No	
Restartable	Yes	
Iterator activity	No	
Supported on	All servers IBM i server Windows server Local (standalone) mode	

**ID** An identifier to uniquely identify this Activity.

**Description** This should describe the Activity.

Status	Active or Inactive. Activities cannot be used in a processing sequence while they are in inactive status – the processing sequence will end in error if it attempts to use an activity that has inactive status.
Activity Processor	Specifies the name of the re-useable part that implements this Activity. For a supplied Activity, this specifies a re-useable part that is shipped with LANSA Composer. If you define your own custom Activities, you must supply your own Activity Processor that implements the custom processing and you must name it here.
Supplied Activity	Yes or No. You cannot change this value. <b>Yes</b> indicates Activities that are supplied with LANSA Composer. <b>No</b> indicates Activities that you define yourself.

ind ed
0
nce it is
ng
0
o to
0
ver d

uncheck the boxes to indicate which servers this activity does support.

## 2.1.3 Activity Parameters

This tab lists the inbound and outbound parameters that have been defined for the Activity. Parameters provide the variable information necessary for an Activity to complete its work. They also provide the communication between different Activities and Transformations that are orchestrated together in a single Processing Sequence. Thus, the output (outbound parameters) from one Activity might provide the input (inbound parameters) for the next.

	Direction	Required or Optional	Parameter Name	Parameter Description	Parameter Class	Parameter Is a List?	Default Value	Bave
1	Outbound		DEMO_BATCHNUM	Batch number	General	No		
					85			<u>A</u> dd
								Remove
								Move up
								Move down

For each parameter, you can specify the following information:

Direction	Inbound, outbound or both. This specifies whether the Activity Processor requires this parameter as input (using the GetVariable method) or will set its value as output (using the PutVariable method).
Required or Optional	This applies only to inbound parameters and its value provides a visual indication, when using the Activity in the <i>Processing sequence editor</i> , of whether the parameter is required.
Parameter Name	Specifies a name for the parameter. By default, this will become the name of the corresponding variable in the variable pool for a Processing Sequence that uses this Activity, though the name can be overridden in the <i>Processing sequence editor</i> .
Parameter Description	This should describe the purpose or use of the parameter.
Parameter Class	You can choose a class for the parameter from the options shown in the drop-down list. The parameter class does not limit or validate the values that can be entered for the parameter, but if you choose a class, other than the default <i>General</i> , LANSA Composer will provide appropriate browsing support for the

	parameter value when entered through the LANSA Composer
	client software. For example, if you choose <i>Trading Partner</i> ,
	LANSA Composer will automatically support browsing and
	selecting from a list of defined Trading Partners. Or if you
	choose <i>File path</i> , LANSA Composer will automatically provide
	support for browsing and selecting from files in the server file
	system.
Parameter Is a List?	Yes or No. If the Activity Processor references this parameter as a list (by using multiple and indexed references to it), you should set this value to Yes. This provides a visual indication in the <i>Processing sequence editor</i> that the associated variable is a list.
Default Value	If required, you can specify a default value for the parameter.

# 2.1.4 Activity Groups

This tab lists the available Activity groups – the groups to which this Activity has been assigned have a checkmark next to their name. Activity groups are used to group related Activities so they can be easily located. You can define your own groups in addition to or instead of the groups supplied with LANSA Composer.

📄 Details 🛛 😤 Para	ameters 🛛 🙆 Groups 🔄 🔁 Run history 🛛 🔍 Where used 🛛 🛄	Notes 🛛 📀 Audit 🛛
Name DESIGN EMAIL	Description Design, test and debug Email	Save Refresh
FILE_MANAGE	File management	
FTP	FTP transport activities	
I HTTP	HTTP transport activities	
MESSAGING	Messaging transport activities	
PROCESS	Processing	
TRANSFORM	Transformations	
TRANSPORT	Transport	
TUT	Tutorials	
VARIABLES	Variable manipulation	
ZIP	Zip activities	

Here you can assign Activities to Activity groups or remove them from Activity groups. You can do this both for supplied Activities and for custom Activities that you define yourself.

To add an Activity to an Activity group, check the box next to the desired group. To remove an Activity from an Activity group, uncheck the box. Remember to click *Save* before switching elsewhere in LANSA Composer.

# 2.2 Activities by Group

LANSA Composer is supplied with a range of ready-to-use Activities that perform transport, file management and other tasks.

These Activities are supplied complete, including, of course, the Activity Processors. However you may need to create one or more Configurations for use with the transport Activities. Refer to each Activity's description for details.

The supplied Activities should not be changed. Future upgrades to LANSA Composer may completely replace the supplied Activities, overwriting any changes you may make. If you wish to adapt the supplied Activities for your own purposes, you should create your own copy of the Activity definition and the Activity Processor and adapt them to your needs.

Refer to the list of 2.2.17 All Supplied Activities

or

refer to the following quick reference lists of functionally-related activities:

2.2.1 Design, Test and Debug	2.2.6 Iterator activities	2.2.11 Terminal Server Activities
2.2.2 Email	2.2.7 Messaging Transport	2.2.12 Transaction Document Processing
2.2.3 File Management	2.2.8 Processing	2.2.13 Transformations
2.2.4 FTP Transport Activities	2.2.9 Spooled File Management	2.2.14 Transport
2.2.5 HTTP Transport Activities	2.2.10 SQL Database Activities	2.2.15 Variable manipulation

#### 2.2.16 Zip Activities

Some Activities from previous versions of LANSA Composer have been deprecated in this version. Refer to 2.2.18 Deprecated Activities for information on deprecated Activities and suggested alternatives.

# 2.2.1 Design, Test and Debug

Activity ID	Description
FOR_EACH_VAR	Iterate for each processing sequence variable
LOGLIST	Creates entries in the processing sequence log that contain a variable list's values.
LOGVARIABLE	Creates an entry in the processing sequence log that contains a variable's value.
NULL	This activity does nothing but can be useful as a placeholder during the design of your processing sequence.

# 2.2.2 Email

Activity ID	Description
MAIL_RECEIVE	Retrieves a single email message from the mail server.
MAIL_RECEIVEALL	Retrieves multiple email messages for a user from the mail server.
MAIL_SEND	Sends an email by SMTP.
SMS_SEND	Send an SMS message to a mobile number using an email service provided by a third-party SMS provider

# 2.2.3 File Management

Activity ID	Description
BASE64_DECODE	Base64 decode the contents of a specified file.
BASE64_ENCODE	Base64 encode the contents of a specified file.
COPY_FILE	Copies a file from one directory to another using the file names, source directory and the target directory.
DELETE_FILE	Deletes a flat file from a directory using the provided file name and full path.
DIRECTORY_LIST	Lists the contents of a directory.
FOR_EACH_TXTLIN	Iterate for each line in a text file.
FOR_EACH_OBJECT	Iterate for each object in an IBM i server library
HASH_FILE	Generate a hash value for a specified file
LOAD_PSVSET	Load processing sequence variables from a PSV file.
MOVE_FILE	Moves a file from one directory to another using the file names, source directory and the target directory.
PATHMAKE	Constructs a file path from folder path and file name.
PATHSPLIT	Splits a file path into its constituent components.
QUERY_CCSID	Returns the IBM i CCSID for a specified file
RENAME_FILE	Renames a file.
SAVE_PSVSET	Save processing sequence variables to a PSV file.
SYSTEM_COMMAND	Executes an operating system command.
TEXT_SUBSTITUTE	Reads a skeleton text file and replaces %%parm.parmeter%% references in the text with the current parameter value and writes the expanded text file.

WAIT_FILESREADY	Cache operations for files waiting for "ready"
WATCH_DIRECTORY	Watch a directory for new or changed files
XML_SPLIT	Split XML document file.
XML_QUERY	Query value(s) in an XML document file.
XML_VALIDATE	Validate an XML document file.
XSL_TRANSFORM	Executes an XSL transformation
ZIP_DIRECTORIES	Zips a list of directories and places the contents in a zip archive.
ZIP_FILES	Zips a list of files and places them in a zip archive.
ZIP_LIST	Lists the contents of a zip archive.
ZIP_UNZIP	Lists the contents of a zip archive.

# 2.2.4 FTP Transport Activities

Activity ID	Description
FTP_COMMANDLIST	Connects to the FTP host specified in the FTP Configuration and then executes FTP commands from a command list file.
FTP_DIRLIST	List all the available files in a remote directory as specified by the FTP configuration
FTP_INBOUND	Retrieves a list of files from a remote host using FTP.
FTP_OUTBOUND	Transfers files from the local machine to a remote host by FTP.
FTP_SCRIPT	Execute an FTP script using the native IBM i5/OS FTP client

# 2.2.5 HTTP Transport Activities

Activity ID	Description
HTTP_GET	Retrieves a document from an HTTP server.
HTTP_INBOUND	Handles an inbound HTTP message and saves it to a specific directory.
HTTP_POST	Sends data to an HTTP server and/or receives response data from the server.

# 2.2.6 Iterator activities

Activity ID	Description
FOR_EACH_CSVROW	Iterate for each row in a CSV file
FOR_EACH_INDEX	Iterate for each index in a range
FOR_EACH_OBJECT	Iterate for each object in an IBM i server library
FOR_EACH_TXDOCO	Iterate for each pending outbound transaction doc
FOR_EACH_TXDOCT	Iterate for each transaction set (message) registered for a transaction document
FOR_EACH_TXTLIN	Iterate for each line in a text file
FOR_EACH_VAR	Iterate for each processing sequence variable
LOAD_PSVSET	Load processing sequence variables from a PSV file.
WATCH_DIRECTORY	Watch a directory for new or changed files
WATCH_DTAQ	Watch a data queue for new entries
WATCH_MSGQ	Watch a message queue for new messages

# 2.2.7 Messaging Transport

Activity ID	Description
DTAQ_CLEAR	Clear a named data queue
DTAQ_RECEIVE	Receive data from a named data queue
DTAQ_SEND	Send data to a named data queue
MSG_RECEIVE	Receives a message from a supported message brokering system such as IBM MQ Series.
MSG_SEND	Sends a file through a supported message brokering system such as IBM MQ Series.
MSGQ_RECEIVE	Receive a message from a message queue
MSGQ_SEND	Send a message to a message queue
WATCH_DTAQ	Watch a data queue for new entries
WATCH_MSGQ	Watch a message queue for new messages

# 2.2.8 Processing

Activity ID	Description
BLANKCONCAT	Concatenates one or more strings, removing trailing blanks and inserting a single blank between each string.
CALCULATE	Perform a simple arithmetic calculation.
CALL_3GL	Calls an RPG, COBOL or other 3GL program on IBM i servers.
CALL_FUNCTION	Calls a LANSA function.
CALL_JAVA	Calls a Java program.
CLEARLIST	Clears a variable list.
CLEARVARIABLE	Clears a variable.
COMPOSER_RUN	Run a LANSA Composer Processing Sequence
CONCAT	Concatenates one or more strings, removing trailing blanks from each.
COUNTLIST	Counts the entries in a variable list.
FIND_TPMAP	Find linked transformation map(s) for a trading partner
GET_DTAARA	Read a value from a data area
JSM_RECLAIM	Reclaim LANSA Integrator JSM resources.
JSM_SCRIPT	Execute a LANSA Integrator JSM script.
LOAD_PSVSET	Load processing sequence variables from a PSV file.
LOGLIST	Creates entries in the processing sequence log that contain a variable list's values.
LOGUSERINFO	Creates an impromptu entry in the processing sequence log.
LOGVARIABLE	Creates an entry in the processing sequence log that
contains a variable's value.

LOWERCASE	Converts uppercase characters to lowercase.
NEXTNUMBER	Generates the next number in a series.
NOTIFYEVENT	Trigger an event notification
PUT_DTAARA	Write a value to a data area
RANDOMNUMBER	Generates a pseudo-random number.
SAVE_PSVSET	Save processing sequence variables to a PSV file.
SLEEP	Suspend processing for a specified time interval
SUBSTITUTE	Formats an input string with variable substitution.
SUBSTITUTE_VAR	Formats an input string with variable substitution.
SUBSTRING	Extracts a portion of a string.
UNIQUEID	Generate a unique identifier.
UPPERCASE	Converts lowercase characters to uppercase.
WAIT_FILESREADY	Cache operations for files waiting for "ready"

# 2.2.9 Spooled File Management

Activity ID	Description
DELETE_SPLF	Deletes an IBM i spool file.
LAST_SPLF	Retrieve identity of the last spooled file created in the current job
MOVE_SPLF	Moves an IBM i spool file to a specified output queue.
SPLF_LIST	Lists IBM i spool files in an output queue that meet specified selection criteria.
SPLF_TO_PDF	Converts an IBM i spool file to a PDF document.
SPLF_TO_TEXT	Converts an IBM i spool file to a text document

# 2.2.10 SQL Database Activities

SQL_CALL	Execute an SQL stored procedure
SQL_CALLQRYCSV	Query database using an SQL stored procedure to CSV
SQL_CALLQUERY	Query database using an SQL stored procedure
SQL_COMMIT	Commit a database transaction using SQL
SQL_CONNECT	Connect to database using SQL
SQL_DISCONNECT	Disconnect from database using SQL
SQL_PARAMS	Set parameter values for SQL operation
SQL_PARAMSCSV	Set parameter values for SQL operation from CSV
SQL_QUERY	Query database using SQL
SQL_QUERYTOCSV	Query database using SQL to output CSV file
SQL_ROLLBACK	Rollback a database transaction using SQL
SQL_UPDATE	Update database using SQL

# 2.2.11 Terminal Server Activities

Activity ID	Description
TS_CAPTURE	Capture aXes 5250 terminal session screen image
TS_CONNECT	Connect to aXes 5250 terminal session
TS_DISCONNECT	Disconnect from aXes 5250 terminal session
TS_EXECUTE	Execute an aXes terminal operations script
TS_GET	Get aXes terminal session attributes
TS_GETBYNAME	Get aXes 5250 terminal session value by name
TS_GETBYPOS	Get aXes 5250 terminal session value by position
TS_GETFIELD	Get aXes 5250 terminal session field attributes
TS_SEND	Send data to aXes 5250 terminal session
TS_SETBYNAME	Set aXes 5250 terminal session value by name
TS_SETBYPOS	Set aXes 5250 terminal session value by position
TS_SETCURSOR	Set aXes 5250 terminal session cursor position

# 2.2.12 Transaction Document Processing

Activity ID	Description
DISCOVER_DOC	Discover the document type for a document file
DISCOVER_EDI	Discover, catalogue and validate EDI document file
DISCOVER_MAP	Discover attributes associated with map
DISCOVER_XML	Discover the root element name of an XML document
EDI_SPLIT	Split EDI document file
FOR_EACH_TXDOCO	Iterate for each pending outbound transaction doc
FOR_EACH_TXDOCT	Iterate for each transaction set (message) registered for a transaction document
TXDOC_ALLOCCTRL	Allocate transaction document control number(s)
TXDOC_EXPORT	Export transaction data to staging files
TXDOC_IMPORT	Import transaction data from staging files
TXDOC_KEYS	Retrieve staging file keys for a pending outbound transaction document
TXDOC_REGISTER	Register transaction document
TXDOC_REGOUTBND	Register pending outbound transaction document
TXDOC_REGOUTEDI	Register pending outbound EDIFACT transaction document
TXDOC_REGOUTX12	Register pending outbound EDI X12 transaction document
TXDOC_STATUS	Update transaction document status
XML_SPLIT	Split XML document file.
XML_QUERY	Query value(s) in an XML document file.
XML_VALIDATE	Validate an XML document file.

# 2.2.13 Transformations

Activity ID	Description
BASE64_DECODE	Base64 decode the contents of a specified file.
BASE64_ENCODE	Base64 encode the contents of a specified file.
FIND_TPMAP	Find linked transformation map(s) for a trading partner
TRANSFORM	Runs a transformation map (used as an alternative to the Transform processing sequence directive where the identifier of the transformation map to run is variable and determined at run-time)

XSL\_TRANSFORM Executes an XSL transformation

# 2.2.14 Transport

Activity ID	Description
FTP_COMMANDLIST	Connects to the FTP host specified in the FTP Configuration and then executes FTP commands from a command list file.
FTP_INBOUND	Retrieves a list of files from a remote host using FTP.
FTP_OUTBOUND	Transfers files from the local machine to a remote host by FTP.
FTP_SCRIPT	Execute an FTP script using the native IBM i5/OS FTP client
HTTP_GET	Retrieves a document from an HTTP server.
HTTP_INBOUND	Handles an inbound HTTP message and saves it to a specific directory.
HTTP_POST	Sends data to an HTTP server and/or receives response data from the server
MAIL_RECEIVE	Retrieves a single email message from the mail server.
MAIL_RECEIVEALL	Retrieves multiple email messages for a user from the mail server.
MAIL_SEND	Sends an email by SMTP.
MSG_RECEIVE	Receives a message from a supported message brokering system such as IBM MQ Series.
MSG_SEND	Sends a file through a supported message brokering system such as IBM MQ Series.
SMS_SEND	Send an SMS message to a mobile number using an email service provided by a third-party SMS provider

# 2.2.15 Variable manipulation

Activity ID	Description
BLANKCONCAT	Concatenates one or more strings, removing trailing blanks and inserting a single blank between each string.
CALCULATE	Perform a simple arithmetic calculation.
CLEARLIST	Clears a variable list.
CLEARVARIABLE	Clears a variable.
CONCAT	Concatenates one or more strings, removing trailing blanks from each.
COUNTLIST	Counts the entries in a variable list.
FOR_EACH_VAR	Iterate for each processing sequence variable
LOAD_PSVSET	Load processing sequence variables from a PSV file.
LOGLIST	Creates entries in the processing sequence log that contain a variable list's values.
LOGVARIABLE	Creates an entry in the processing sequence log that contains a variable's value.
LOWERCASE	Converts uppercase characters to lowercase.
NEXTNUMBER	Generates the next number in a series.
PATHMAKE	Constructs a file path from folder path and file name.
PATHSPLIT	Splits a file path into its constituent components.
RANDOMNUMBER	Generates a pseudo-random number.
SAVE_PSVSET	Save processing sequence variables to a PSV file.
SORT_LISTS	Sort one or more variable lists in "parallel"
SUBSTITUTE	Formats an input string with variable substitution.
SUBSTITUTE_VAR	Formats an input string with variable substitution.

SUBSTRING	Extracts a portion of a string.
UNIQUEID	Generate a unique identifier.
UPPERCASE	Converts lowercase characters to uppercase.

# 2.2.16 Zip Activities

Activity ID	Description
ZIP_DIRECTORIES	Zips a list of directories and places the contents in a zip archive.
ZIP_FILES	Zips a list of files and places them in a zip archive.
ZIP_LIST	Lists the contents of a zip archive.
ZIP_UNZIP	Lists the contents of a zip archive.

## 2.2.17 All Supplied Activities

This is a list of all the supplied Activities. There is a quick reference list of Activities in 2.2 Activities by Group.

Activity ID	Description
BASE64_DECODE	Base64 decode the contents of a specified file.
BASE64_ENCODE	Base64 encode the contents of a specified file.
BLANKCONCAT	Concatenates one or more strings, removing trailing blanks and inserting a single blank between each string.
CALCULATE	Perform a simple arithmetic calculation.
CALL_3GL	Calls an RPG, COBOL or other 3GL program on IBM i servers
CALL_FUNCTION	Calls a LANSA function.
CALL_JAVA	Calls a Java program.
CLEARLIST	Clears a variable list.
CLEARVARIABLE	Clears a variable.
COMPOSER_RUN	Run a LANSA Composer Processing Sequence
CONCAT	Concatenates one or more strings, removing trailing blanks from each.
COPY_FILE	Copies a file from one directory to another using the file names, source directory and the target directory.
COUNTLIST	Counts the entries in a variable list.
DELETE_FILE	Deletes a flat file from a directory using the provided file name and full path.
DELETE_SPLF	Deletes an IBM i spool file.
DIRECTORY_LIST	Lists the contents of a directory.

DISCOVER_DOC	Discover the document type for a document file
DISCOVER_EDI	Discover, catalogue and validate EDI document file
DISCOVER_MAP	Discover attributes associated with map
DISCOVER_XML	Discover the root element name of an XML document
DTAQ_CLEAR	Clear a named data queue
DTAQ_RECEIVE	Receive data from a named data queue
DTAQ_SEND	Send data to a named data queue
EDI_SPLIT	Split EDI document file
FIND_TPMAP	Find linked transformation map(s) for a trading partner
FOR_EACH_CSVROW	Iterate for each row in a CSV file
FOR_EACH_INDEX	Iterate for each index in a range
FOR_EACH_OBJECT	Iterate for each object in an IBM i server library
FOR_EACH_TXDOCO	Iterate for each pending outbound transaction doc
FOR_EACH_TXDOCT	Iterate for each transaction set (message) registered for a transaction document
FOR_EACH_TXTLIN	Iterate for each line in a text file
FOR_EACH_VAR	Iterate for each processing sequence variable
FTP_COMMANDLIST	Connects to the FTP host specified in the FTP Configuration and then executes FTP commands from a command list file.
FTP_DIRLIST	Connect to the host specified via FTP and then returns a list of files available in the remote directory.
FTP_INBOUND	Retrieves a list of files from a remote host using FTP.
FTP_OUTBOUND	Transfers files from the local machine to a remote

host by FTP.

Execute an FTP script using the native IBM i5/OS FTP client
Read a value from a data area
Generate a hash value for a specified file
Retrieves a document from an HTTP server.
Handles an inbound HTTP message and saves it to a specific directory.
Sends data to an HTTP server and/or receives response data from the server
Reclaim LANSA Integrator JSM resources.
Execute a LANSA Integrator JSM script
Retrieve identity of the last spooled file created in the current job
Loads processing sequence variables from one or more PSV sets into a PSV file.
Creates entries in the processing sequence log that contain a variable list's values.
Creates an impromptu entry in the processing sequence log.
Creates an entry in the processing sequence log that contains a variable's value.
Converts uppercase characters to lowercase.
Retrieves a single email message from the mail server.
Retrieves multiple email messages for a user from the mail server.
Sends an email by SMTP.

MOVE_FILE	Moves a file from one directory to another using the file names, source directory and the target directory.
MOVE_SPLF	Moves an IBM i spool file to a specified output queue.
MSG_RECEIVE	Receives a message from a supported message brokering system such as IBM MQ Series.
MSG_SEND	Sends a file through a supported message brokering system such as IBM MQ Series.
MSGQ_RECEIVE	Receive a message from a message queue
MSGQ_SEND	Send a message to a message queue
NEXTNUMBER	Generates the next number in a series.
NOTIFYEVENT	Trigger an event notification
NULL	This activity does nothing but can be useful as a placeholder during the design of your processing sequence.
PATHMAKE	Constructs a file path from folder path and file name.
PATHSPLIT	Splits a file path into its constituent components.
PUT_DTAARA	Write a value to a data area
QUERY_CCSID	Returns the IBM i CCSID for a specified file
RANDOMNUMBER	Generate a pseudo-random number
RENAME_FILE	Renames a file.
SAVE_PSVSET	Saves processing sequence variables and their values to a PSV file.
SLEEP	Suspend processing for a specified time interval
SMS_SEND	Send an SMS message to a mobile number using an email service provided by a third-party SMS

provider

SORT\_LISTS Sort one or more variable lists in "parallel"

- SPLF\_LISTLists IBM i spool files in an output queue that meet<br/>specified selection criteria.
- **SPLF\_TO\_PDF** Converts an IBM i spool file to a PDF document.
- **SPLF\_TO\_TEXT** Converts an IBM i spool file to a text document
- SQL\_CALL Execute an SQL stored procedure
- SQL\_CALLQRYCSV Query database using an SQL stored procedure to CSV
- SQL\_CALLQUERY Query database using an SQL stored procedure
  - Commit a database transaction using SQL
- SQL\_CONNECT Connect to database using SQL
- SQL\_DISCONNECT Disconnect from database using SQL
- SQL\_PARAMS Set parameter values for SQL operation
- SQL\_PARAMSCSV Set parameter values for SQL operation from CSV
  - Query database using SQL
- SQL\_QUERYTOCSV Query database using SQL to output CSV file
- SQL\_ROLLBACK Rollback a database transaction using SQL
- SQL\_UPDATE Update database using SQL
- **SUBSTITUTE** Formats an input string with variable substitution.
- **SUBSTITUTE\_VAR** Formats an input string with variable substitution.
- **SUBSTRING** Extracts a portion of a string.
- **SYSTEM\_COMMAND** Executes an operating system command.
- TEXT\_SUBSTITUTE

SQL\_COMMIT

SQL\_QUERY

Reads a skeleton text file and replaces %%parm.parmeter%% references in the text with the current parameter value and writes the expanded text file.

TRANSFORM	Runs a transformation map (used as an alternative to the Transform processing sequence directive where the identifier of the transformation map to run is variable and determined at run-time)
TS_CAPTURE	Capture aXes 5250 terminal session screen image
TS_CONNECT	Connect to aXes 5250 terminal session
TS_DISCONNECT	Disconnect from aXes 5250 terminal session
TS_EXECUTE	Execute an aXes terminal operations script
TS_GET	Get aXes terminal session attributes
TS_GETBYNAME	Get aXes 5250 terminal session value by name
TS_GETBYPOS	Get aXes 5250 terminal session value by position
TS_GETFIELD	Get aXes 5250 terminal session field attributes
TS_SEND	Send data to aXes 5250 terminal session
TS_SETBYNAME	Set aXes 5250 terminal session value by name
TS_SETBYPOS	Set aXes 5250 terminal session value by position
TS_SETCURSOR	Set aXes 5250 terminal session cursor position
TXDOC_ALLOCCTRL	Allocate transaction document control number(s)
TXDOC_EXPORT	Export transaction data to staging files
TXDOC_IMPORT	Import transaction data from staging files
TXDOC_KEYS	Retrieve staging file keys for a pending outbound transaction document
TXDOC_REGISTER	Register transaction document
TXDOC_REGOUTBND	Register pending outbound transaction document
TXDOC_REGOUTEDI	Register pending outbound EDIFACT transaction document
TXDOC_REGOUTX12	Register pending outbound EDI X12 transaction

	document
TXDOC_STATUS	Update transaction document status
UNIQUEID	Generate a unique identifier.
UPPERCASE	Converts lowercase characters to uppercase.
WAIT_FILESREADY	Cache operations for files waiting for "ready"
WATCH_DIRECTORY	Watch a directory for new or changed files
WATCH_DTAQ	Watch a data queue for new entries
WATCH_MSGQ	Watch a message queue for new messages
XML_SPLIT	Split XML document file.
XML_QUERY	Query value(s) in an XML document file.
XML_VALIDATE	Validate an XML document file.
XSL_TRANSFORM	Executes an XSL transformation.
ZIP_DIRECTORIES	Zips a list of directories and places the contents in a zip archive.
ZIP_FILES	Zips a list of files and places them in a zip archive.
ZIP_LIST	Lists the contents of a zip archive.
ZIP_UNZIP	Lists the contents of a zip archive.

## BASE64\_DECODE

This activity will base64 decode the contents of a specified file and write the decoded content to another file.

Base64 encoding is commonly used when there is a need to encode binary data that needs to be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 encoding is commonly used in a number of applications including email via MIME, and storing complex data in XML.

#### **INPUT Parameters:**

#### BASE64SOURCE : Required

This parameter must specify the full path and file name of the file whose contents are to be base64 decoded.

#### BASE64TARGET : Optional

This parameter may specify the full path and file name for the file to which the decoded file contents are to be written. The default, if not specified, is the special value \*AUTO.

If the special value \*AUTO is used, the activity will use the path and file name specified in the BASE64SOURCE parameter but replacing the file extension with an extension of '.decode'. For example, if you specify /myfolder/myfile.dat in the BASE64SOURCE parameter, then the activity will write the decoded contents to the file /myfolder/myfile.decode.

#### BASE64REPLACE : Optional

This parameter specifies what the activity should do if the output file already exists. The default value \*NO will cause the activity to end in error if the output file already exists. Alternatively you can specify \*YES to cause the existing file to be replaced.

#### OUTPUT Parameters: BASE64DECODED :

Upon successful completion, this parameter will contain the actual path and file name of the output file containing the decoded contents of the file specified by the BASE64SOURCE parameter.

## BASE64\_ENCODE

This activity will base64 encode the contents of a specified file and write the encoded content to another file.

Base64 encoding is commonly used when there is a need to encode binary data that needs to be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 encoding is commonly used in a number of applications including email via MIME, and storing complex data in XML.

#### **INPUT Parameters:**

#### BASE64SOURCE : Required

This parameter must specify the full path and file name of the file whose contents are to be base64 encoded.

#### BASE64TARGET : Optional

This parameter may specify the full path and file name for the file to which the encoded file contents are to be written. The default, if not specified, is the special value \*AUTO.

If the special value \*AUTO is used, the activity will use the path and file name specified in the BASE64SOURCE parameter but replacing the file extension with an extension of '.base64'. For example, if you specify */myfolder/myfile.dat* in the BASE64SOURCE parameter, then the activity will write the encoded contents to the file */myfolder/myfile.base64*.

#### BASE64FOLD : Optional

This parameter specifies whether the activity should fold the encoded contents onto separate lines. The default value \*NO will not insert any line breaks in the encoded contents. Alternatively you can specify \*YES to cause the activity to insert a carriage-return and line-feed (CRLF) after every 76 characters of encoded output.

#### BASE64REPLACE : Optional

This parameter specifies what the activity should do if the output file already exists. The default value \*NO will cause the activity to end in error if the output file already exists. Alternatively you can specify \*YES to cause the existing file to be replaced.

#### OUTPUT Parameters: BASE64ENCODED :

Upon successful completion, this parameter will contain the actual path and file name of the output file containing the encoded contents of the file specified by the BASE64SOURCE parameter.

## BLANKCONCAT

This activity concatenates one or more strings, removing trailing blanks and inserting a single blank between each input string. At least two and up to 9 input strings may be specified, either as literals or variables or a mixture of both.

#### **INPUT Parameters:**

#### **STRINGIN1** : Required

This parameter specifies the first string to be trimmed and concatenated.

#### **STRINGIN2** : Required

This parameter specifies the second string to be trimmed and concatenated.

STRINGIN3STRINGIN4 STRINGIN5 STRINGIN6 STRINGIN7 STRINGIN8 STRINGIN9 : Optional

These parameters may be used to specify further strings to be blank concatenated. If used, they must be specified contiguously (the activity stops looking after the first parameter whose value is not specified).

#### **OUTPUT Parameters:**

STRINGOUT :

Upon completion, this parameter will contain the blank concatenated string.

# CALCULATE

This activity performs a simple arithmetic calculation, given an operator and two operands. It is intended for simple calculations, usually using integer values. The calculation for the modulo (remainder) operator assumes integer values are specified.

All calculations are performed using a precision of 30,9. If any inputs or intermediate or final results exceed the number of significant or decimal digits, then loss of precision will result without warning or error.

Note: Intensive calculations, especially when relating to application data (as opposed to variables that control the process orchestration) should be done in compiled code units (such as custom activities, Java or IBM i 3GL programs or LANSA functions) that are called from the processing sequence.

#### **INPUT Parameters:**

#### NUMBERIN1 : Required

This parameter specifies the first operand for the calculation. The value specified must be numeric.

#### **OPERATOR** : Required

This parameter must specify the arithmetic operator to be used for the calculation. You must specify one of the following values:

- + (addition)
- (subtraction)
- \* (multiplication)
- / (division)

% (modulo, remainder)

#### NUMBERIN2 : Required

This parameter specifies the second operand for the calculation. The value specified must be numeric. Further, for operators / and % (division and modulo operations), the value cannot be zero.

#### **OUTPUT Parameters:**

NUMBEROUT :

Upon successful completion, this output parameter contains the result of the specified calculation.

# CALL\_3GL

This activity calls a program. The program may be an RPG, COBOL or Control Language program or written using some other 3GL supported on IBM i. This activity is only supported on IBM i servers.

The activity can pass and receive up to 9 parameters, according to the requirements of the specified program. All parameters are passed as A(256). The called program should treat the parameters as character data and must not attempt to address more than 256 bytes for each parameter.

It is the user's responsibility:

- to ensure that the program is available in the library list at execution time

- to ensure that the executing job has the necessary authorities to make the call

- to specify at least the number of parameters expected by the called program

- to ensure that the called program is appropriate for the purpose.

#### **INPUT Parameters:**

**PGM** : Required

This parameter specifies the name of the program to call.

LIB : Optional

This parameter can specify the name of the library containing the program to call. If it is not specified, the processing sequence will use the library list to locate the program.

#### **INPUT and OUTPUT Parameters:**

PARM01PARM02 PARM03 PARM04 PARM05 PARM06 PARM07 PARM08 PARM09 : Optional

These parameters can be used to pass and receive up to nine parameters to/from the called program. The parameters are passed and received as character variables of length 256. The activity will pass and receive the number of parameters (up to nine) for which you specify values or alternate variable names. If used, they must be specified contiguously (the activity

stops looking after the first parameter whose value is not specified).

# CALL\_FUNCTION

This activity calls a named LANSA function. The LANSA function may be contained in another LANSA configuration on the same or a different (\*) server. The activity can pass and receive up to seven values via the LANSA exchange list. All parameters are passed as A(256).

(\* Note a LANSA Composer Remote Request Server license is required for the remote server in order to call a LANSA function on a different server system.) It is the user's responsibility:

- To ensure that the function is available in the necessary LANSA system and partition
- To ensure that the executing job (\*) has the necessary authorities to make the call
- To ensure that the called function is appropriate for the purpose.

(\*) Depending on the request, the executing job will usually be a job submitted by the LANSA Composer Request Server.

Further important information about the CALL\_FUNCTION activity is provided under the following headings later in these notes:

LANSA programming considerations for the called function

Calling a function in the same system and partition as LANSA Composer

Calling a function in another LANSA system or partition

Function calls executed through the LANSA Composer Request Server

You should also refer to:

2.3.10 LANSA System Configuration

Appendix F. The LANSA Composer Request Server

## **INPUT Parameters:**

#### LANSACONFIG : Optional

This parameter specifies the name of a LANSA system configuration that identifies the LANSA system and partition containing the function to be called and the connection details, if required, to connect to the server system on which the LANSA system resides. The partition containing the function to be called must be a multilingual partition.

## **PROCESS** : Optional

This parameter specifies the name of the process containing the function to

call. If not specified, the activity assumes \*DIRECT. If \*DIRECT is specified or assumed, the function must be defined with FUNCTION OPTIONS(\*DIRECT). On Windows servers the process name (not \*DIRECT) must be specified for requests that will be processed through the request server.

#### **FUNCTION** : Required

This parameter specifies the name of the function to call. The function name must always be specified.

#### SYNCHRONOUS : Optional

This parameter specifies whether the activity waits for the function call to complete. It defaults to YES, which means the activity does wait. If any other value is specified, the activity posts the function call request and completes immediately. Note that the activity can only receive values returned from the called function (in the EXCH01 ... EXCH07 input/output parameters) if this parameter is YES.

#### SYNCHTIMEOUT : Optional

This parameter specifies the number of seconds the activity waits for a synchronous call to complete when executed through the request server. If not specified, a default of 30 seconds is used. If the timeout is exceeded, the activity ends with an error. Note the timeout ONLY applies to requests executed through the request server.

#### **EXPIRES** : Optional

On IBM i servers only, this parameter specifies the number of seconds, after it is posted to the request server, that the request remains effective. If more than the specified interval has elapsed before the request server begins to process the request, the request server will consider the request to have expired and will not process it. If not specified, a default of zero (0) is used, which means that no expiry applies to the request. Note that the expiry ONLY applies to requests executed through the request server for IBM i servers. No expiry applies when running on Windows servers.

#### LANSASYS : Optional (deprecated)

This parameter specifies the name of the program library for the LANSA system containing the function to call. If not specified, the activity assumes the same LANSA system as is executing LANSA Composer. This parameter is only used on IBM i servers and is provided for backwards compatibility. On Windows servers (and for new solutions on IBM i servers) you should

specify a LANSA system configuration name in the LANSACONFIG parameter. If a LANSA system configuration is named in the LANSACONFIG parameter, the value of this parameter is not used.

#### **PARTITION** : Optional (deprecated)

This parameter specifies the name of the partition containing the function to call. If not specified, the activity assumes the same partition in which LANSA Composer is executing. This parameter is only used on IBM i servers and is provided for backwards compatibility. On Windows servers (and for new solutions on IBM i servers) you should specify a LANSA system configuration name in the LANSACONFIG parameter. If a LANSA system configuration is named in the LANSACONFIG parameter, the value of this parameter is not used. The partition containing the function to be called must be a multilingual partition.

#### **INPUT and OUTPUT Parameters:**

EXCH01EXCH02 EXCH03 EXCH04 EXCH05 EXCH06 EXCH07 : Optional

These parameters can be used to pass and receive up to seven values to/from the called function via the LANSA exchange list. The parameters are placed on and received from the exchange list as character variables of length 256 using the variable names EXCH01 ... EXCH07.

The called function must also use the variables names EXCH01 ... EXCH07 in order to receive the parameter values. If the called function needs to return values via these variables, it must execute the EXCHANGE command at the appropriate point.

The activity will place on and receive from the exchange list the number of parameters (up to seven) for which you specify values or alternate variable names. If used, they must be specified contiguously (the activity stops looking after the first parameter whose value is not specified).

Note that the activity can only receive values returned from the called function when executed synchronously - see the description of the SYNCHRONOUS parameter.

Refer to the description of the EXCHANGE RDML command in the LANSA product documentation for further information on exchanging

information via the exchange list.

# LANSA programming considerations for the called function

The following LANSA programming considerations apply to the way in which the called function must be defined:

- If \*DIRECT is specified or assumed for the PROCESS parameter, the function must be defined with FUNCTION OPTIONS(\*DIRECT)
- Called functions may be RDML or they may be fully RDMLX enabled.
- The partition containing the function to be called must be a multi-lingual partition.
- The function must define fields EXCH01 ... EXCH07 in order to receive values (via the exchange list) that are specified in the corresponding CALL\_FUNCTION activity parameters.
- The function must use the EXCHANGE command with fields EXCH01 ... EXCH07 in order to return values (via the exchange list) for variables that are specified in the corresponding CALL\_FUNCTION activity parameters.
- On IBM i servers only, if the CALL\_FUNCTION activity will execute via the LANSA Composer request server, position 487 of LANSA data area DC@A01 must be set to 'Y' before compiling the function. If this condition is not met, the called function will not correctly receive or return the EXCH01 ... EXCH07 variable values.

Refer to LANSA documentation for more information on LANSA features referred to above.

Depending on all the requirements, these considerations may sometimes necessitate that your programming staff write functions specifically for the purpose. However, even if this is necessary, the functions can frequently be simple "stub" functions that call other existing functions in the LANSA application.

## Calling a function in the same system and partition as LANSA Composer

If you want to call a function that is available in the same LANSA system and partition as LANSA Composer, you only need to specify the function name in the FUNCTION parameter, and any values to be exchanged with the function in the EXCH01 ... EXCH07 parameters. You do not need to specify values for the LANSACONFIG, PROCESS or SYNCHRONOUS parameters (nor the deprecated LANSASYS and PARTITION parameters).

Providing (1) \*DIRECT is specified or assumed for the PROCESS parameter, and (2) YES is specified or assumed for the SYNCHRONOUS parameter, the function call will be performed directly from the activity processor, in the same job and context. When the CALL\_FUNCTION activity runs in this mode, the value of the SYNCHTIMEOUT parameter is ignored.

Functions called in this way must be defined with FUNCTION OPTIONS(\*DIRECT).

# Calling a function in another LANSA system or partition

You can call a function that is available in a different LANSA system and/or partition than LANSA Composer on the same or a different server system. To do this, you should:

- a)create a LANSA system configuration in LANSA Composer that identifies the LANSA system and partition you wish to use, and
- b)specify the LANSA system configuration name on the LANSACONFIG parameter.

Such requests offer greater flexibility, including the ability to call the required function through a specified process (instead of with \*DIRECT) and synchronously or asynchronously processing the request. These types of function calls will execute through the LANSA Composer Request Server, and some special considerations apply.

# Function calls executed through the LANSA Composer Request Server

The CALL\_FUNCTION activity will seek to execute the function call through the LANSA Composer request server if any of the following are true:

• You specify a LANSA system configuration name using the LANSACONFIG parameter

(OR on IBM i servers only, the values specified for the LANSASYS and PARTITION parameters identify a LANSA system and/or partition that is not the same as for the LANSA Composer system.)

- You specify a process name (other than \*DIRECT) in the PROCESS parameter.
- You specify a value other than YES for the SYNCHRONOUS parameter.

When executed this way, the function call executes in another process or job (the request server). LANSA Composer and the request server process or job

communicate cooperatively to execute the request and return the results. Some special considerations apply to this mode of execution, including considerations related to:

- User profiles, authorities and execution environment
- IBM i work management (jobs and subsystems)
- The way in which the called function must be compiled

You should refer to Appendix F (The LANSA Composer Request Server) in the LANSA Composer guide for detailed information about considerations for requests executed through the LANSA Composer Request Server.

# CALL\_JAVA

This activity calls a named Java program, and can pass up to 9 command-line parameter (or argument) values. It cannot receive parameter values back from the Java program, however.

It is the user's responsibility to ensure the program and all required supporting components are available in the default or specified classpath at execution time and that the executing job has the necessary authorities to make the call.

The CALL\_JAVA activity will use a Java environment with system default execution attributes. There is no means by which you can specify or override the Java VM version or other parameters such as those affecting JVM garbage collection.

Depending on the operating environment, load and capacity, the overhead of starting a JVM for each execution of the CALL\_JAVA activity may become excessive in some applications. The CALL\_JAVA activity is intended mostly for relatively infrequent non-intensive use. If your application calls for intensive use (for example, executing a Java program repeatedly in a tight loop), then you should consider and test the performance characteristics before commiting to this activity as a solution.

#### **INPUT Parameters:**

#### **CLASS** : Required

This parameter specifies the name of the Java class to be run. The class name may be qualified by one or more package names. Each package name must be followed by a period. For example 'pkg1.pkg2.myClass'. You may specify the special value '\*VERSION' to have Java run with the -version switch and list the JVM version information in the processing sequence log (when run with sufficient logging in effect). Alternatively, you may specify the name of a jar file in which the startup class is indicated by the Main-Class specified in the manifest.

#### **CLASSPATH** : Optional

This parameter can specify the path used to locate Java classes. Directories are separated by colons. If not specified, the classpath will be determined by the operating environment.

#### PARM01PARM02 PARM03

#### PARM04 PARM05 PARM06 PARM07 PARM08 PARM09 : Optional

These parameters can be used to pass up to nine command-line parameter (or argument) values to the called Java program. The activity will pass the number of parameters (up to nine) for which you specify values or alternate variable names. If used, they must be specified contiguously (the activity stops looking after the first parameter whose value is not specified).

#### **OUTPUT Parameters:**

There are no output parameters.

## CLEARLIST

This activity clears the variable list specified in the parameter.

#### **INPUT Parameters:**

**LIST** : Required

This parameter specifies the variable list to be cleared.

You should always specify a variable, not a literal value, for this parameter. The variable list's values are not used by this activity - rather the variable list specified is cleared of all values.

#### **OUTPUT Parameters:**

There are no output parameters.

## CLEARVARIABLE

This activity clears the value of the variable specified in the parameter.

For most purposes, this is functionally equivalent to assigning an empty value to the variable using an ASSIGN processing sequence directive.

#### **INPUT Parameters:**

#### VARIABLE : Required

This parameter specifies the variable whose value is to be cleared.

You should always specify a variable, not a literal value, for this parameter. The variable's value is not used by this activity - rather the variable is cleared of its value.

#### **OUTPUT Parameters:**

There are no output parameters.
# COMPOSER\_RUN

This activity runs a named LANSA Composer processing sequence. It can pass up to five named parameter values to the processing sequence. The processing sequence to be run does not have to be and usually is not in the same LANSA Composer configuration or even on the same server (\*) system as the activity that initiates the run. The processing sequence can be run synchronously or asynchronously.

(\* Note a LANSA Composer Remote Request Server license is required for the remote server in order to run a Processing Sequence on a different server system.)

#### NOTE:

The COMPOSER\_RUN activity will run the specified processing sequence through the LANSA Composer Request Server. This means that the processing sequence runs in another process or job (the request server). LANSA Composer and the request server communicate cooperatively to execute the request and return the results.

As a consequence, some particular considerations apply, including considerations related to:

- User profiles, authorities and the execution environment
- IBM i work management (jobs and subsystems)

Refer to 2.3.10 LANSA System Configuration for information about the LANSA system configurations that are used in conjunction with this activity and with the CALL\_FUNCTION activity.

Refer to Appendix F. The LANSA Composer Request Server in the LANSA Composer guide for detailed information about considerations for requests executed through the LANSA Composer request server.

## **INPUT Parameters:**

## LANSACONFIG : Optional

This parameter specifies the name of a LANSA system configuration that identifies the LANSA system and partition containing the processing sequence to run and the connection details, if required, to connect to the server system on which the LANSA system resides.. This parameter is required on Windows servers and recommended on IBM i servers.

**PSEQ** : Required

This parameter identifies the processing sequence to run. Either the external identifier (name) or the internal identifier (as shown on the Audit tab) may be specified.

## SYNCHRONOUS : Optional

This parameter specifies whether the activity waits for the processing sequence run to complete. It defaults to YES, which means the activity does wait. If any other value is specified, the activity posts the processing sequence run request and completes immediately.

## SYNCHTIMEOUT : Optional

This parameter specifies the number of seconds the activity waits for a synchronous run to complete. If not specified, a default of 30 seconds is used. If the timeout is exceeded, the activity ends with an error.

## **EXPIRES** : Optional

On IBM i servers only, this parameter specifies the number of seconds, after it is posted to the request server, that the request remains effective. If more than the specified interval has elapsed before the request server begins to process the request, the request server will consider the request to have expired and will not process it. If not specified, a default of zero (0) is used, which means that no expiry applies to the request. No expiry applies when running on Windows servers.

#### PARMNAME01, PARMVALUE01PARMNAME02, PARMVALUE02 PARMNAME03, PARMVALUE03 PARMNAME04, PARMVALUE04 PARMNAME05, PARMVALUE05 : Optional

These parameters can be used to pass up to five processing sequence parameter name and value pairs to the processing sequence.

- PARMNAME01 (... PARMNAME05) should contain the parameter name as defined in the processing sequence to be run
- PARMVALUE01 (... PARMVALUE05) should contain the value that is to be passed to the processing sequence for the corresponding parameter name. The maximum value length that can be passed is 200.

If used, they must be specified contiguously (the activity stops looking after the first parameter name/value pair that is not specified).

## LANSASYS : Optional (deprecated)

This parameter specifies the name of the program library for the LANSA system containing the processing sequence to run. If not specified or if

\*CURRENT is specified, the activity assumes the same LANSA system as is executing LANSA Composer. This parameter is only used on IBM i servers and is provided for backwards compatibility. On Windows servers (and for new solutions on IBM i servers) you should specify a LANSA system configuration name in the LANSACONFIG parameter. If a LANSA system configuration is named in the LANSACONFIG parameter, the value of this parameter is not used.

#### **PARTITION** : Optional (deprecated)

This parameter specifies the name of the partition containing the processing sequence to run. If not specified or if \*CURRENT is specified, the activity assumes the same partition in which LANSA Composer is executing. This parameter is only used on IBM i servers and is provided for backwards compatibility. On Windows servers (and for new solutions on IBM i servers) you should specify a LANSA system configuration name in the LANSACONFIG parameter. If a LANSA system configuration is named in the LANSACONFIG parameter, the value of this parameter is not used.

#### **OUTPUT Parameters:**

There are no output parameters.

# CONCAT

This activity concatenates one or more strings, removing trailing blanks from each string as it does so. At least two and up to 9 input strings may be specified, either as literals or variables or a mixture of both.

#### **INPUT Parameters:**

#### **STRINGIN1** : Required

This parameter specifies the first string to be trimmed and concatenated.

#### STRINGIN2 : Required

This parameter specifies the second string to be trimmed and concatenated.

STRINGIN3STRINGIN4 STRINGIN5 STRINGIN6 STRINGIN7 STRINGIN8 STRINGIN9 : Optional

These parameters may be used to specify further strings to be concatenated. If used, they must be specified contiguously (the activity stops looking after the first parameter whose value is not specified).

#### OUTPUT Parameters: STRINGOUT :

Upon completion, this parameter will contain the concatenated string.

# COPY\_FILE

This Activity will copy a file from one directory to another.

When executed on a Windows platform a system copy command is executed.

When executed on an IBM i platform an i5/OS CPY command is executed. Note that files in the integrated file system are copied – if you want to copy objects in the QSYS file system, you must use IFS file system notation to do so.

When executed on an IBM i platform, this activity is capable of performing data conversion between different CCSIDs during the copy operation. For example, to convert the data in a text file from its original encoding to UTF-8 you could use the parameter values TOCCSID(1208) DTAFMT(\*TEXT).

## **INPUT Parameters:**

## **FROMFULLNAME** : Required

This parameter should contain the full path and name of the file to be copied

For example: Windows C:\dir1\myfile.txt

IBM i /indir/myfile.txt

## **TODIRECTORY** : Optional (\*)

This parameter may contain the full directory path of the destination directory. If specified, the file will be copied to this new location using the same name and extension as the original file. If not specified, you must provide a value for the TOFULLNAME parameter.

## **TOFULLNAME** : Optional (\*)

This parameter may specify the path of the directory the object is to be copied to AND the new name of the object. If specified, the file will be copied to this new location using the new file name and extension specified. If not specified, you must provide a value for the TODIRECTORY parameter.

## FROMCCSID : Optional, IBM i servers only

This parameter may specify the method for obtaining the coded character set identifier (CCSID) for the source of the copy operation. This CCSID will be used for data conversion, if requested.

This parameter corresponds directly to the FROMCCSID parameter of the IBM i CPY command and you may use any values that are defined for the CPY command. At IBM i V7R1 the possible values include \*OBJ, \*PCASCII, \*JOBCCSID or a CCSID value in the range 1-65533. Refer to

the IBM documentation for the CPY command for more information.

The special (default) value \*DEFAULT specifies that the FROMCCSID parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

TOCCSID : Optional, IBM i servers only

This parameter may specify the data coded character set identifier (CCSID) for the target of the copy operation.

This parameter corresponds directly to the TOCCSID parameter of the IBM i CPY command and you may use any values that are defined for the CPY command. At IBM i V7R1 the possible values include \*OBJ, \*CALC, \*PCASCII, \*STDASCII, \*JOBCCSID or a CCSID value in the range 1-65533. Refer to the IBM documentation for the CPY command for more information.

The special (default) value \*DEFAULT specifies that the TOCCSID parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

DTAFMT : Optional, IBM i servers only

This parameter may specify the format of the data in the file to be copied..

This parameter corresponds directly to the DTAFMT parameter of the IBM i CPY command and you may use any values that are defined for the CPY command. At IBM i V7R1 the possible values are \*TEXT and \*BINARY. Refer to the IBM documentation for the CPY command for more information.

The special (default) value \*DEFAULT specifies that the DTAFMT parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

**AUT** : Optional, IBM i servers only

This parameter may specify the method used to assign authority information to copied objects.

This parameter corresponds directly to the AUT parameter of the IBM i CPY command and you may use any values that are defined for the CPY command. At IBM i V7R1 the possible values are \*OBJ, \*INDIR and \*INDIROBJ. Refer to the IBM documentation for the CPY command for more information.

The special (default) value \*DEFAULT specifies that the AUT parameter will not be specified. Note that this parameter applies only when LANSA

Composer is running on an IBM i server.

#### **OUTPUT Parameters: FULLNAMEOUT**

Upon successful completion, this parameter will contain the full path and file name of the resulting destination file.

# COUNTLIST

This activity counts the entries in the variable list specified in the parameter.

### **INPUT Parameters:**

**LIST** : Required

This parameter specifies the variable list to be counted.

You should always specify a variable, not a literal value, for this parameter.

#### **OUTPUT Parameters:**

COUNT :

Upon successful completion, this output parameter contains the number of entries in the specified variable list.

## DELETE\_FILE

This Activity will delete a flat file from a directory.

The full path and name of the file must be provided.

When executed on a Windows platform a system del command is executed.

When executed on an IBM i platform an i5/OS DEL command is executed. Note that files in the integrated file system are deleted – if you want to delete objects in the QSYS file system, you must use IFS file system notation to do so.

## **INPUT Parameters:**

## FILENAME : Required

This parameter should contain the full path and name of the file to be deleted

For example: Windows c:\mydir\myfile.txt

IBM i /dirabc/file.txt

#### **OUTPUT Parameters:**

There are no output parameters.

## DELETE\_SPLF

This activity will delete a spool file on an IBM i server. It is only supported on IBM i servers.

The input parameters must fully identify the spool file to be deleted. You must specify the SPLFNUMBER parameter if there is more than one spool file of the name specified by the SPLFNAME parameter for the job specified by the first three parameters

#### **INPUT Parameters:**

#### **JOBNAME** : Required

This parameter specifies the name of the job that created the spool file to be deleted.

#### JOBUSER : Required

This parameter specifies the user profile name of the job that created the spool file to be deleted.

#### JOBNUMBER : Required

This parameter specifies the job number of the job that created the spool file to be deleted.

#### **SPLFNAME** : Required

This parameter specifies the name of the spool file to be deleted.

#### SPLFNUMBER : Optional

This parameter specifies the number of the spool file to be deleted and is only required if there is more than one spool file of the name specified by the SPLFNAME parameter for the job specified by the first three parameters. If not specified, a default special value of \*ONLY is used. You may specify special values \*ONLY or \*LAST. Otherwise specify the number of the job's spooled file that is to be deleted.

#### **OUTPUT Parameters:**

There are no output parameters.

# DIRECTORY\_LIST

This activity will list the contents of a directory. Using this activity you can list either the files or the directories contained in the specified directory. You can also specify whether the list should include contents of nested sub-directories contained in the specified directory.

Optionally, files (but not sub-directories) may be filtered using the FILENAME and/or EXTENSION parameters. If neither parameter is specified, all available files will be included in the file list (when the content requested is \*FILES).

Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

#### **INPUT Parameters:**

**DIRECTORY** : Required

This parameter should contain the full path of the directory whose contents are to be listed.

For example: Windows C:\mydirectory

IBM i /orders/January

**CONTENT** : Optional

This parameter specifies whether files or directories contained in the specified directory are to be listed. You may specify one of the following values:

\*FILES (to list the files) \*DIRECTORIES (to list the directories)

(The alias value \*FOLDERS may also be used as an alternative to \*DIRECTORIES)

If this parameter is not specified, a default of \*FILES is assumed.

## MAXDEPTH : Optional

This parameter specifies whether the content of sub-directories of the specified directory should be included, and, if so, the nested depth to which this should apply. You may specify one of the following values:

1 (lists only the files or directories that are immediate children of the specified directory)

*n* (list files or directories to the nesting depth specified by the integer value) \*MAX (lists files or directories contained in the specified directory and all

## descendant directories)

If this parameter is not specified, a default of 1 (one) is assumed.

Note that in order to avoid unintended "runaway" processing, the special value \*MAX actually limits the nesting depth to 9 (nine), which would be reasonable for most circumstances. If you wish to process directory contents to a nesting depth greater than this, then you may do so by explicitly specifying an appropriate integer value.

CAUTION: Use this parameter with care. It is very easy to inadvertently make a request much larger than you might have anticipated. Specifying a value greather than 1 (one) in conjunction with a directory that contains many files or sub-directories may lead to an unexpectedly long processing time for the activity and/or to processing limits being exceeded.

## FILENAME : Optional

This parameter may be specified when you wish to list files that match a specified file name and/or extension pattern. This parameter only applies when the content requested (in the CONTENT parameter) is \*FILES.

The filter matching is not case sensitive. For example a value of 'abc\*.txt' will result in only files with an extension of .TXT or .txt and with names beginning with 'ABC' or 'abc' being returned in the file list.

When constructing the file name mask, you may use the following placeholder characters:

Will match any single character at the specified position of the name or ? extension of the file name

Will generically match any/all remaining characters in the name or

\* extension component of the file name

Within either the name or extension component of the file name mask, any characters after the first '\*' are not effective and will be ignored. If there is no extension component, then files of any (or no) extension will be matched. Examples:

- \*.\* Will list all files (equivalent to omiting this parameter value)
- ab\*.\* Will list all files whose names begin with 'ab' and with any (or no) file extension

??? Will list all files whose names contain 'd' in the 4th position and that d\*.csv have an extension of 'csv'

\*.xm\* Will list all files whose file extension begins with 'xm'

## **EXTENSION** : Optional

This parameter may be specified when you wish to list only files with a certain extension. This parameter only applies when the content requested (in the CONTENT parameter) is \*FILES.

The filter matching is not case sensitive and does not require the '.' prefix. For example a value of 'xml' will result in only files with an extension of .XML or .xml being returned in the file list.

This parameter provides better performance than the FILENAME parameter alone if you only want to select files by file extension. (This is because it is implemented at the LANSA Integrator XMLFileService level, rather than as a post-processing step). You may specify both the FILENAME and EXTENSION parameters - the value of the EXTENSION parameter is applied first and matching files are then tested aainst the FILENAME mask, if provided.

## **ORDERBY** : Optional

This parameter allows you to specify the order in which the files will be listed in the FILELIST output parameter. You may specify one value from each of the following two groups (each value should be separated by at least one space):

- 1. \*NONE | \*NAME | \*MODIFIED
- 2. \*ASCEND | \*DESCEND

You must specify quote marks around the value(s) to distinguish them from built-in variable names.

The values are described further below. If you specify more than one value from each group, the last-specified value is effective. If you do not specify this parameter, the default values is \*NONE, meaning that no explicit ordering is performed (in this case the actual order is undefined by LANSA Composer but may be subject to operating environment factors).

## 1. \*NONE | \*NAME | \*MODIFIED

These values specify the file attribute that is used to order the list of files. Specify \*NAME to have the list ordered by file path. Specify \*MODIFIED to have the list ordered by the date last modified for each file.

```
2. *ASCEND | *DESCEND
```

These values specify the ordering to apply. Specify \*ASCEND to list the files in ascending order or \*DESCEND for descending order. If not specified, the default is \*ASCEND.

For example, specify '\*MODIFIED \*DESCEND' to have the files listed in descending order of the last-modified date.

NOTE: The ORDERBY parameter applies whether the content requested (in the CONTENT parameter) is \*FILES or \*DIRECTORIES. If the MAXDEPTH parameter specifes a value greater than 1 (one), ORDERBY affects the order of files or directories *within* each directory listed.

#### **OUTPUT Parameters: FILELIST**

Specifies the name of a list variable that, upon successful completion, will contain a list of the files or directories that match the request.

The list will contain the full path and file or directory name. If you specify a value other than \*NONE for the ORDERBY parameter, the list entries will be ordered as specified (within each sub-directory, if applicable).

For example: Windows C:\mydirectory\file1.txt

IBM i /orders/January/ord01.xml

# DISCOVER\_DOC

This activity determines the document type (as defined in LANSA Composer's *Document types*) corresponding to a specified transaction document file.

#### **INPUT Parameters:**

#### **DOCFILE** : Required

This parameter specifies the path and name of the transaction document file.

## DOCTYPEGROUP : Optional

This parameter may specify the name of a document type group that will limit the document type definitions against which the specified transaction document file is matched. The document type groups provide a further means for you to qualify the document discovery process according to the context in which it is running.

#### **OUTPUT Parameters: DOCTYPE** :

Upon successful completion, this parameter will contain the name of the document type that applies to the specified transaction document file. If a matching document type is not found, this parameter will contain the special value '\*UNKNOWN'.

## DOCPSEQ :

Upon successful completion, this parameter will contain the name of the processing sequence associated with the document type (if specified for the document type). Your processing sequence can invoke the processing sequence using the Processing Sequence directive (the processing Sequence directive allows you to specify a processing sequence variable that contains the processing sequence identifier).

# DISCOVER\_EDI

This activity processes an EDI transaction document file to discover more information about its contents. If the document has already been registered in the transaction document register, the activity will update the register with information extracted from the document.

The functions of this activity include:

- determining the EDI document standard, version and the transaction type;
- matching the EDI transaction document to a LANSA Composer trading partner definition, and updating the transaction document register accordingly;
- populating the trading partner agreements (interchanges), groups and transactions (messages) sections of the transaction document register with initial information about the contents of the EDI transaction document;
- optionally validating the EDI transaction(s) (if specified for the matching trading partner, and if a SEF file is specified for the matching LANSA Composer document standard and version).

### **INPUT Parameters:**

#### **EDINUMBER** : Required

This parameter specifies the transaction document envelope number for the transaction document. This number is normally assigned by the TXDOC\_REGISTER activity.

If the document has not been registered, you can specify the special value '\*NONE'. If you specify '\*NONE', the activity will not attempt to use or update the transaction document register with information gleaned from the document.

## **EDIFILE** : Required

This parameter specifies the path and name of the EDI transaction document file.

#### **EDIDIR** : Required

This parameter specifies the direction (I=inbound, O=outbound) for the document exchange.

#### OUTPUT Parameters: EDISTD :

Upon successful completion, this parameter will contain the document

standard id (as defined in LANSA Composer's Document types) that applies to the EDI transaction document.

## EDIVER :

Upon successful completion, this parameter will contain the document standard version id (as defined in LANSA Composer's Document types) that applies to the EDI transaction document.

## EDITYPE :

Upon successful completion, this parameter will contain the EDI transaction type (eg: 850 or INVOIC) of the EDI transaction(s) contained in the EDI transaction document.

## EDIPRODTEST :

Upon successful completion, this parameter will contain P if the EDI transaction is a production transaction or T if it is a test transaction.

## EDIVALID:

Upon successful completion, this parameter indicates whether the transaction is deemed to be valid (Y) or not (N), if the transaction was validated according to the LANSA Composer settings in effect for the applicable trading partner and document standard.

## EDIMATCHTP :

Upon successful completion, this parameter will contain the identifier of the trading partner that matches the EDI transaction document.

# DISCOVER\_MAP

This activity retrieves attributes (including many data interchange attributes) associated with a specified transformation map definition.

### **INPUT Parameters:**

#### **MAPID** : Required

This parameter specifies the identifier of the transformation map whose attributes are to be retrieved. You may specify the special value \*TRANSFORM to retrieve attributes for the current transformation map (the transformation map identified by the \*TRANSFORM built-in value).

## **OUTPUT Parameters:**

#### MAPTYPE :

Upon successful completion, this parameter will contain the transformation map type code value that applies to the specified transformation map.

## DOCTYPE :

Upon successful completion, this parameter will contain the name of the document type (if any) associated with the specified transformation map.

## **DIRECTION**:

Upon successful completion, this parameter will contain the map direction (I, O) that applies to the specified transformation map.

## **STANDARD**:

Upon successful completion, this parameter will contain the name of the document standard that applies to the specified transformation map.

## **VERSION**:

Upon successful completion, this parameter will contain the name of the document standard version that applies to the specified transformation map.

## **TRANSACTIONID** :

Upon successful completion, this parameter will contain the transaction type that applies to the specified transformation map.

## **DOCEXTN** :

Upon successful completion, this parameter will contain the file extension associated with the document type.

## **DOCPSEQIN** :

Upon successful completion, this parameter will contain the identifier of the

inbound processing sequence associated with the document type. Your processing sequence can invoke the processing sequence using the Processing Sequence directive.

## **DOCPSEQOUT** :

Upon successful completion, this parameter will contain the identifier of the outbound processing sequence associated with the document type. Your processing sequence can invoke the processing sequence using the Processing Sequence directive.

# DISCOVER\_XML

This activity interrogates an XML document to discover the type of information it contains. A processing sequence might use the output from this activity to determine how to process different types of XML documents - for example, what type of transformation to run for them.

If the activity fails to load or parse the document, it will set an error return code.

For performance reasons, this activity performs simplified parsing of the XML document. It does NOT load the entire document and cannot assert that the XML document is well-formed or valid. In some cases, it may return, without error, a root element name for invalid or badlyformed XML documents.

#### **INPUT Parameters:**

#### **XMLFILE** : Required

This parameter specifies the path and name of the XML document.

#### **OUTPUT Parameters:** XMLROOT :

Upon successful completion, this parameter will contain the name of the root element of the specified XML document. The namespace prefix, if present, has been removed.

## **XMLROOTUPPER** :

Upon successful completion, this parameter will contain the same value as the XMLROOT output parameter, but with lowercase characters translated to uppercase.

**Note:** XML tags are strictly case-sensitive. The uppercase version of the document root element name is provided for your convenience, but it is your responsibility to ensure that using this converted name is valid and appropriate for your application.

## XMLNAMESPACE :

Upon successful completion, this parameter will contain the namespace URI associated with the root element, if any.

## XMLNAMESPACEUPPER :

Upon successful completion, this parameter will contain the same value as the XMLNAMESPACE output parameter, but with lowercase characters

translated to uppercase.

**Note:** XML tags are strictly case-sensitive. The uppercase version of the namespace URI is provided for your convenience, but it is your responsibility to ensure that using this converted name is valid and appropriate for your application.

# DTAQ\_CLEAR

This activity clears a named data queue.

Refer also to the companion DTAQ\_RECEIVE, DTAQ\_SEND and WATCH\_DTAQ activities that allows you to send and receive data to and from a named data queue.

Note: Data queues are persistent objects but their data content can be lost or corrupted during a system failure. Backup, recovery and maintenance of data queues used in your LANSA Composer solution is your responsibility – you should make provision for this in your implementation plans.

## **IBM i Platform Notes**

Data queues are a native operating system feature on the IBM i server. The full range of capabilities of this activity may only be available on IBM i servers.

(The implementation of this activity on an IBM i server uses the QCLRDTAQ system API.)

Refer to the description of the DTAQ\_SEND activity for more complete IBM i Server Platform Notes relating to the suite of DTAQ activities. Refer to IBM i documentation concerning data queues for further information about the operating system features upon which this activity depends.

## Windows Server Platform Notes

Although data queues are a native feature of the IBM i server, this activity and the companion DTAQ\_RECEIVE, DTAQ\_SEND and WATCH\_DTAQ activities are available on Windows servers too, but with a reduced feature set and with other restrictions and caveats.

The following notes and restrictions apply to using this activity on a Windows server:

- The DTAQLIB parameter is not used and its value will be ignored. (You may use the DTAQPATH parameter to specify a location for the data queue, if required.)
- Data queues are stored in pairs of files in the Windows server file system with .EDQ and .LDQ file extensions. The .EDQ file contains the data queue definition and entries. The .LDQ files are used to logically lock the data queue during receive operations. The file names are determined by the LANSA run-time support used by this activity. This activity clears the data

queue by deleting these files.

- The location of the .EDQ and .LDQ data queue storage files is determined by the value of the DTAQPATH parameter. Refer to that parameter description for further information.
- Refer to the description of the DTAQ\_SEND activity for more complete Windows Server Platform Notes relating to the suite of DTAQ activities.

## **INPUT Parameters:**

#### **DTAQ** : Required

This parameter must specify the name of the data queue that is to be cleared.

## **DTAQLIB**: Optional

When running on IBM i servers, this parameter may specify the name of the library containing the data queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named data queue.

When running on a Windows server, this parameter is not used and its value is ignored. You may use the DTAQPATH parameter to specify the location for the data queue, if required.

#### **DTAQPATH:** Optional

When running on Windows servers, this parameter may specify the path to the directory in which the data queue .EDQ and .LDQ files are stored. If not specified, the default location is determined by the LANSA run-time according to the effective value of the DPTH= X\_RUN parameter.

When running on an IBM i server, this parameter is not used and its value is ignored. You may use the DTAQLIB parameter to specify the location for the data queue, if required.

#### **OUTPUT Parameters:**

There are no output parameters

# DTAQ\_RECEIVE

This activity receives data from a named data queue.

This activity treats data received from a data queue as character data. Using this activity to interact with applications that send or receive binary data to or from the data queue is not supported.

When the activity receives an entry from the named data queue, the data content (and possibly sender information) is available through the output parameters of the activity.

When no data is available to be received (after waiting for the interval specified by the DTAWAIT parameter, if used), the activity ends normally but the output parameters are not filled. The value of the RCVCOUNT output parameter may be used to determine whether an entry was received.

Refer also to the companion DTAQ\_CLEAR, DTAQ\_SEND and WATCH\_DTAQ activities that allows you to send and receive data to and from and to perform other operations on a named data queue.

Note: Data queues are persistent objects but their data content can be lost or corrupted during a system failure. Backup, recovery and maintenance of data queues used in your LANSA Composer solution is your responsibility – you should make provision for this in your implementation plans.

## **IBM i Platform Notes**

Data queues are a native operating system feature on the IBM i server. The full range of capabilities of this activity is only available on IBM i servers.

(The implementation of this activity on an IBM i server uses the QRCVDTAQ system API.)

Use of keyed data queues with this activity is not presently supported.

Refer to the description of the DTAQ\_SEND activity for more complete IBM i Server Platform Notes relating to the suite of DTAQ activities. Refer to IBM i documentation concerning data queues for further information about the operating system features upon which this activity depends.

## Windows Server Platform Notes

Although data queues are a native feature of the IBM i server, this activity and the companion DTAQ\_CLEAR, DTAQ\_SEND and WATCH\_DTAQ activities

are available on Windows servers too, but with a reduced feature set and with other restrictions and caveats.

(The implementation of this activity on a Windows server uses the emulated data queue support in the LANSA run-time, using the RCV\_FROM\_DATA\_QUEUE built-in function.)

The following notes and restrictions apply to using this activity on a Windows server:

- The DTAQLIB parameter is not used and its value will be ignored. (You may use the DTAQPATH parameter to specify a location for the data queue, if required.)
- Data queues are stored in pairs of files in the Windows server file system with .EDQ and .LDQ file extensions. The .EDQ file contains the data queue definition and entries. The .LDQ files are used to logically lock the data queue during receive operations. The file names are determined by the LANSA run-time support used by this activity. The files can be manually cleared or deleted by deleting the associated files. Alternatively the DTAQ\_CLEAR activity may be used for this purpose.
- The location of the .EDQ and .LDQ data queue storage files is determined by the value of the DTAQPATH parameter. Refer to that parameter description for further information.
- Refer to the description of the DTAQ\_SEND activity for more complete Windows Server Platform Notes relating to the suite of DTAQ activities.

NOTE: On Windows servers, the specific implementation used means that the activity is unable to handle certain types of exceptions that may be considered unexceptional in other circumstances. In such cases the processing sequence containing the activity may simply end abnormally without logging diagnostic information in the Processing Sequence log. This behavior differs from most other LANSA Composer activities and from the IBM i implementation of this activity.

You should be aware of this possibility and be prepared to diagnose such issues in other ways. For example, the x\_err.log generated by the LANSA run-time may assist with diagnosing such conditions.

#### **INPUT Parameters:**

## **DTAQ** : Required

This parameter must specify the name of the data queue from which the data is to be received.

## **DTAQLIB**: Optional

When running on IBM i servers, this parameter may specify the name of the library containing the data queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named data queue.

When running on a Windows server, this parameter is not used and its value is ignored. You may use the DTAQPATH parameter to specify the location for the data queue, if required.

## **DTAQPATH**: Optional

When running on Windows servers, this parameter may specify the path to the directory in which the data queue .EDQ and .LDQ files are stored. If not specified, the default location is determined by the LANSA run-time according to the effective value of the DPTH= X\_RUN parameter.

When running on an IBM i server, this parameter is not used and its value is ignored. You may use the DTAQLIB parameter to specify the location for the data queue, if required.

## **DTAWAIT**: Optional

Specifies the length of time (in seconds) that the activity will wait for data to arrive on the data queue if no data is immediately available when the activity is processed.

The default value is 0 (zero) which means that the activity does not wait for the arrival of an entry if one is not immediately available when the activity is processed.

You may also specify the special value \*MAX which means that the activity waits indefinitely for data to arrive. You should understand that this value may cause the activity and hence the processing sequence in which it is contained to stall indefinitely if no data is sent to the data queue. Other than by the arrival of data, such a processing sequence run may only be terminated by terminating the job, the subsystem in which the job is running or the system. If you wish a process to wait indefinitely for data to arrive on a data queue, the WATCH\_DTAQ activity may provide a more suitable solution.

## **DTARMV**: Optional

Specifies whether the data received by the activity is removed from the data

queue. The default is \*YES, which means the data is removed. You may specify \*NO to cause the data not to be removed from the data queue. If you do so, the data can be received again by this activity in the same or another Processing Sequence.

## **DTAQELEN**: Optional

On an IBM i server, this parameter is not required and its value will be ignored.

On a Windows server, this parameter is required and it must specify the length of the data queue entries. The activity supports data lengths between 1 and 1024 inclusive.

On a Windows server, the first send (DTAQ\_SEND) operation for a data queue will automatically create the data queue and will set its entry length. The length specified for subsequent operations to the same data queue must match the length specified on the initial operation that created the data queue. (On Windows servers, LANSA Composer will actually use the next greatest multiple of 256 – for example, if you specify 700 for this parameter, LANSA Composer will actually use the value 768.)

#### OUTPUT Parameters: RCVCOUNT:

The received entry count. If data is received successfully, the value will be one (1). Otherwise the value will be zero (0). The activity ends normally when no message is available to receive after waiting for the interval specified by the DTAWAIT parameter.

## **RCVDATALEN**:

If data is received, this parameter contains the length of the data received.

(On a Windows server, the activity cannot determine the actual data length received, and the value returned in this parameter is the same as the value provided in the DATAQELEN input parameter.)

#### RCVDATARCVDATA02 RCVDATA03 RCVDATA04:

If data is received, this parameter contains the data. Each parameter receives up to 256 bytes of data. The first parameter, RCVDATA, contains the first up to 256 bytes of data. The remaining parameters are filled only if you are using data queue entries of length greater than 256, 512 and 768 respectively.

## SENDJOB

## SENDUSER SENDJOBNBR:

If data is received, these output parameters may contain the job name, user name and the job number of the job that sent the data to the data queue.

On an IBM i server, these values will be available only if

SENDERID(\*YES) was specified when the data queue was created using the CRTDTAQ (Create Data Queue) command.

On a Windows server, the sending job information is not available and these parameters will not be filled.

# DTAQ\_SEND

This activity sends data to a named data queue.

This activity treats data sent to a data queue as character data. Using this activity to interact with applications that send or receive binary data to or from the data queue is not supported.

Refer also to the companion DTAQ\_RECEIVE, DTAQ\_CLEAR and WATCH\_DTAQ activities that allows you to receive data from and to perform other operations on a named data queue.

Note: Data queues are persistent objects but their data content can be lost or corrupted during a system failure. Backup, recovery and maintenance of data queues used in your LANSA Composer solution is your responsibility – you should make provision for this in your implementation plans.

## **IBM i Platform Notes**

Data queues are a native operating system feature on the IBM i server. The full range of capabilities of this activity are only available on IBM i servers.

(The implementation of this activity on an IBM i server uses the QSNDDTAQ system API.)

Data queues referenced by this activity must already exist on the IBM i server. You can create a data queue on the IBM i server using the CRTDTAQ command. For example, the following command creates a data queue named MYDATAQ in library QGPL that can receive data entries of up to 256 characters:

## CRTDTAQ DTAQ(QGPL/MYDATAQ) MAXLEN(512)

Use of keyed data queues with this activity is not presently supported.

Refer to IBM i documentation concerning data queues for further information about the operating system features upon which this activity depends.

## Windows Server Platform Notes

Although data queues are a native feature of the IBM i server, this activity and the companion DTAQ\_CLEAR, DTAQ\_RECEIVE and WATCH\_DTAQ activities are available on Windows servers too, but with a reduced feature set and with other restrictions and caveats.

(The implementation of this activity on a Windows server uses the emulated

data queue support in the LANSA run-time, using the SND\_TO\_DATA\_QUEUE built-in function.)

The following notes and restrictions apply to using this activity on a Windows server:

- The TODTAQLIB parameter is not used and its value will be ignored. (You may use the TODTAQPATH parameter to specify a location for the data queue, if required.)
- Data queues are automatically created when they are first referenced.
- The first reference not only creates the data queue, but also permanently sets the entry size for the data queue according to the value of the DTALEN parameter.
- Data queues are stored in pairs of files in the Windows server file system with .EDQ and .LDQ file extensions. The .EDQ file contains the data queue definition and entries. The .LDQ files are used to logically lock the data queue during receive operations. The file names are determined by the LANSA run-time support used by this activity. The files can be manually cleared or deleted by deleting the associated files. Alternatively the DTAQ\_CLEAR activity may be used for this purpose.
- The location of the .EDQ and .LDQ data queue storage files is determined by the value of the TODTAQPATH parameter. Refer to that parameter description for further information.
- If you have the Visual LANSA documentation available to you, you can find further information pertinent to the Windows implementation of this and related activities in the description of the SND\_TO\_DATA\_QUEUE built-in function, in the LANSA Technical Reference Guide.

NOTE: On Windows servers, the specific implementation used means that the activity is unable to handle certain types of exceptions that may be considered unexceptional in other circumstances. In such cases the processing sequence containing the activity may simply end abnormally without logging diagnostic information in the Processing Sequence log. This behavior differs from most other LANSA Composer activities and from the IBM i implementation of this activity.

You should be aware of this possibility and be prepared to diagnose such issues in other ways. For example, the x\_err.log generated by the

LANSA run-time may assist with diagnosing such conditions.

## **INPUT Parameters:**

## TODTAQ : Required

This parameter must specify the name of the data queue to which the data is sent.

## TODTAQLIB: Optional

When running on IBM i servers, this parameter may specify the name of the library containing the data queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named data queue.

When running on a Windows server, this parameter is not used and its value is ignored. You may use the TODTAQPATH parameter to specify a location for the data queue, if required.

## TODTAQPATH: Optional

When running on Windows servers, this parameter may specify the path to the directory in which the data queue .EDQ and .LDQ files are to be stored. If not specified, the default location is determined by the LANSA run-time according to the effective value of the DPTH= X\_RUN parameter.

When running on an IBM i server, this parameter is not used and its value is ignored. You may use the TODTAQLIB parameter to specify a location for the data queue, if required.

## **DTALEN:** Required

This parameter must specify the number of characters of data to be sent to the data queue. The activity supports data lengths between 1 and 1024 inclusive.

On an IBM i server, this value must be consistent with the length specified when the data queue was created using the CRTDTAQ command.

On a Windows server, the first send operation for a data queue will automatically create the data queue and will set its entry length. The length specified for subsequent operations to the same data queue must match the length specified on the initial operation that created the data queue. (On Windows servers, LANSA Composer will actually use the next greatest multiple of 256 – for example, if you specify 700 for this parameter, LANSA Composer will actually use the value 768.)

#### SNDDATASNDDATA02 SNDDATA03

#### SNDDATA04: Optional

These parameters specify the data to be sent to the data queue. Each parameter specifies up to 256 bytes of data. The first parameter, SNDDATA is required. The remaining parameters are optional and only need be specified if you are using data queue entries of length greater than 256, 512 and 768 respectively.

#### **OUTPUT Parameters:**

There are no output parameters

# EDI\_SPLIT

This activity can split a composite EDI transaction document file into one or more EDI transaction documents each containing no more than the specified number of transactions. This splitting can be important or necessary for two main reasons concerning further processing of the EDI transaction document:

- later standard LANSA Composer EDI document processing, such as the DISCOVER\_EDI activity is founded on assumptions that the transactions contained in EDI document are for the same trading partner and/or are of the same EDI transaction type;
- when very large EDI transaction documents are expected, splitting them into smaller documents optimises their further processing, especially in any mapping steps.

#### **INPUT Parameters:**

#### **EDIFILE** : Required

This parameter specifies the path and name of the EDI transaction document file to be split.

#### **EDILIMIT** : Optional

This parameter specifies the maximum number of transactions to be contained in each of the resulting split EDI transaction documents. If not specified, a default of 1 (one) is assumed. The value for this parameter can be taken from the value specified for the exchange trading partner associated with the EDI transaction document file using the \*tradingpartner.edi.splitmax built-in variable.

#### EDIIGNORECRLF : Optional

When Y (yes) is specified, carriage return / line feeds in the source EDI transaction document file are ignored when parsing the document. If not specified, a default of N (no) is used. The value for this parameter can be taken from the value specified for the exchange trading partner associated with the EDI transaction document file using the \*tradingpartner edi splitignerecr built in variable

\*tradingpartner.edi.splitignorecr built-in variable.

# OUTPUT Parameters:

## EDISPLITLIST :

Upon successful completion, this parameter will contain a list of the resulting EDI transactions documents. The individual split documents in the list are typically further processed using the LOOP directive.

# FIND\_TPMAP

This activity finds the transformation map(s) linked to the specified trading partner that match the criteria specified. Transformation maps are considered in order of the sequence (within map type) specified when the maps are linked to the trading partner.

Depending on the transformation maps linked to the trading partner and the criteria specified in the parameters to this activity, more than one matching map may be found. Consequently, the output parameter is a list of transformation map identifiers. If you want or intend that only one matching map be found, then you must:

a) link only the appropriate maps to the trading partner

b) ensure that the linked maps have appropriate attributes, including the applicable Data Interchange attributes

c) specify appropriate values for the activity parameters.

If more than one matching map is found, the output parameters MAPID and MAPACK will be treated as a list variable, each containing an entry for each matching transformation map that was found. Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

## **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the identifier of the trading partner whose linked transformation maps are to be considered. If you are using this activity inside a processing sequence loop that is controlled by the \*TRADINGPARTNERS built-in variable list, then you should usually specify the \*TRADINGPARTNER built-in variable for this parameter. You may also use the \*TRADINGPARTNER built-in variable if, in your processing sequence, you have previously explicitly set its value. Otherwise, you should specify a variable or literal that contains the required trading partner ID.

## **PRODTEST** : Optional

Specifies whether the transformation map must be linked to the trading partner for production (P) or test (T) transactions in order to be considered. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **MAPTYPE** : Optional

Specifies the transformation map type code value that is to be matched against the type of transformation maps linked to the trading partner. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **DOCTYPE** : Optional

Specifies the transaction document type identifier that is to be matched against the transaction document types associated with transformation maps linked to the trading partner. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **DIRECTION** : Optional

Specifies the map direction that is to be matched against the map direction associated with transformation maps linked to the trading partner. You should specify 'I' for inbound maps or 'O' for outbound maps. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **STANDARD** : Optional

Specifies the document standard "match agency" value that is to be matched against that specified for the document standard associated with transformation maps linked to the trading partner (for example, you would specify 'X' for X12 maps). If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **VERSION** : Optional

Specifies the document standard "match version" value that is to be matched against that specified for the document standard associated with transformation maps linked to the trading partner (for example, you would specify '004010' for X12 version 004010 maps). If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## TRANSACTIONID : Optional

Specifies the map transaction ID that is to be matched against the map transaction ID associated with transformation maps linked to the trading partner. For example, you might specify '850' for maps that process X12 transaction 850. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

**VERSION** : Optional

Specifies the map standard version that is to be matched against the map standard version associated with transformation maps linked to the trading partner. If not specified, then the selection of matching linked transformation map is not affected by that attribute.

## **OUTPUT Parameters:**

#### MAPCOUNT

Upon completion, this variable will contain a count of the number of matching transformation maps found. (This will be the same as the number of items in the MAPID list).

#### MAPID

Upon successful completion, this parameter will contain a list of the identifiers (names) of transformation maps linked to the specified trading partner that match the specified criteria. In many cases, you might expect one matching map (but this is subject to appropriate entry of map attributes and activity parameter values). In such cases, you simply refer to the first (and assumed to be the only) entry in the list. In other cases, you may expect more than one matching linked transformation map. In that event, you should process the resulting list in this parameter using a LOOP directive in the processing sequence.

Note that you can assign the value(s) contained in this parameter to the \*TRANSFORM built-in variable. You can then directly access further attributes of the map using the supported qualified forms of the \*TRANSFORM built-in variable.

#### MAPACK

Upon successful completion, this parameter will contain a list of the "acknowledgement required" flags corresponding to the matching transformation maps identified in the MAPID parameter list. This value (Y/N) comes from the "Linked Transformation Maps" definition for the trading partner and transformation map and specifies whether the transaction processed by the transformation map for the trading partner requires an acknowledgement to be sent.
# FOR\_EACH\_CSVROW

This is an iterator activity. It will read each row in a CSV file and on each iteration output first 50 column values found in the row. The processing logic nested beneath FOR\_EACH\_CSVROW activity is repeated for each row read.

This activity supports reading up to 999 rows and outputs the value for up to 50 columns. It is not intended for routine processing of large volumes of data. It can be useful however for transferring limited amounts of information between activities, transformation maps and the processing sequence variable pool.

### Note:

This activity will read only CSV file format.

This activity will read only the first 999 rows of a CSV file and the remaining rows are ignored.

This activity will output up to 50 columns of a CSV file and the remaining columns are ignored.

# **INPUT Parameters:**

### **CSVFILEPATH** : Required

This parameter must contain the path of the file to be read.

This activity reads only CSV file format.

eg C:\order.csv

or /orders/order\_jan.csv

### SEPARATOR: Optional

If a non-comma separator is used, this value should contain the separator character. The separator should be 1 character in length and can consist of any character.

### OUTPUT Parameters: CSVCOLUMN1CSVCOLUMN2

## •••

# CSVCOLUMN50

These output parameters will contain the value for the corresponding column for each CSV row read, up to the number of columns present in the data or a maximum of 50 columns.

# FOR\_EACH\_INDEX

This is an iterator activity. It will iterate for each index value in the range specified by the input parameters. The processing logic nested beneath the activity is repeated for each index value.

### **INPUT Parameters:**

### **INDEXFROM** : Optional

This parameter specifies the starting index value. If not specified a default value of 1 (one) is used.

### **INDEXTO** : Required

This parameter specifies the ending index value. Iteration ends when the current index value exceeds the value of this parameter.

### **INDEXSTEP** : Optional

This parameter specifies the value by which the index is incremented for each iteration. If not specified a default value of 1 (one) is used.

# **OUTPUT Parameters:**

### INDEXOUT

This parameter contains the current index value during each iteration.

# FOR\_EACH\_OBJECT

This is an iterator activity that iterates for each object matching specified criteria in a named IBM i server library. The processing sequence directives nested beneath the activity are repeated for each matching object found. This activity is supported only on IBM i servers.

### **INPUT Parameters:**

### LIB : Required

This parameter must specify the name of the library containing the objects to be iterated.

You may also specify one of the special values \*ALL, \*ALLUSR, \*CURLIB, \*LIBL, \*USRLIBL.

You should be very careful if you specify some of these special values that will enumerate more than one library, especially the \*ALL or \*ALLUSR values, as the request may take a considerable time to process.

### **OBJ** : Optional

This parameter specifies which objects in the specified library or libraries are to be iterated. You may use the default value \*ALL or you may specify the name or generic name of the objects required. A generic name is specified as a character string that contains one or more characters followed by an asterisk (\*). For example, the value ABC\* selects all objects whose names begin with the characters 'ABC' that exist in the specified library or libraries and that also match the object type specified by the OBJTYPE parameter.

## **OBJTYPE** : Optional

This parameter specifies the object type of the objects in the specified library or libraries that are to be iterated. You may use the default value \*ALL or you may specify a particular IBM i object type such as \*PGM, \*FILE etc.

# **OUTPUT Parameters: OBJOUT**

Upon each iteration, this parameter will contain the name of the current object.

## LIBOUT

Upon each iteration, this parameter will contain the name of the library containing the current object. In most cases, this will be the same value as specified on the LIB input parameter, unless you used one of the special values on the LIB parameter that do not identify a particular library name

such as \*ALL or \*ALLUSR.

### **OBJTYPEOUT**

Upon each iteration, this parameter will contain the object type of the current object – for example, \*PGM or \*FILE.

### **OBJATTROUT**

Upon each iteration, this parameter will contain the attribute of the current object – for example, an object of type \*FILE might have one of several possible attributes including PF, LF or SAVF.

### **OBJSIZEOUT**

Upon each iteration, this parameter will contain the size of the current object.

### **OBJTEXTOUT**

Upon each iteration, this parameter will contain the text description of the current object.

# FOR\_EACH\_TXDOCO

This is an iterator activity. It will iterate for each pending outbound transaction document in LANSA Composer's document register that matches the criteria specified by the parameter values. For each iteration, it provides the transaction document envelope number in the DOCNUMBER output parameter.

Typically a process using this activity specifies further processing directives nested beneath this iterator activity to extract or export, transform and send the pending outbound transaction document.

## **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the trading partner whose pending outbound transaction documents are to be processed.

### **MAPID** : Required

Specifies the transformation map identifier associated with pending outbound transaction documents to be processed.

### **PRODTEST** : Optional

This parameter specifies whether production (P) or test (T) pending outbound transaction documents are to be processed. If not specified, a default of production (P) is used.

### **DOCTYPE** : Optional

This parameter specifies additional restriction on which pending outbound transaction documents are to be processed. If specified, only documents of a specific type will be processed.

### **DOCSTD** : Optional

This parameter specifies additional restriction on which pending outbound transaction documents are to be processed. If specified, only documents of a specific standard will be processed.

### **DOCSTDVER** : Optional

This parameter specifies additional restriction on which pending outbound transaction documents are to be processed. If specified, only documents registered as of a certain standard version will be processed.

## **DOCCONTENTTYPE** : Optional

This parameter specifies additional restriction on which pending outbound transaction documents are to be processed. If specified, only documents

registered as having a certain content type will be processed.

### DOCDATAKEY01DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 : Optional

These parameters refer to application defined "keys" that may have been specified when douments were registered. If specified, only those documents registered with matching key will be processed.

### OUTPUT Parameters: DOCNUMBER

Upon each iteration, this parameter will contain the transaction document envelope number of a pending outbound transaction document that matched the criteria specified. This number is typically referenced in further processing directives nested beneath this iterator activity to extract or export, transform and send the pending outbound transaction document.

# DOCTYPEOUT

Upon each iteration, this parameter will contain the document type of a pending outbound transaction document that matched the criteria specified.

# DOCSTDOUT

Upon each iteration, this parameter will contain the document standard of a pending outbound transaction document that matched the criteria specified.

# DOCSTDVEROUT

Upon each iteration, this parameter will contain the document standard version of a pending outbound transaction document that matched the criteria specified.

# DOCPRODTESTOUT

Upon each iteration, this parameter will contain P for production usage or T for test usage.

# DOCCONTENTTYPEOUT

Upon each iteration, this parameter will contain the document content type of a pending outbound transaction document that matched the criteria specified.

DOCDATAKEY01OUT DOCDATAKEY02OUT DOCDATAKEY03OUT

### DOCDATAKEY04OUT DOCDATAKEY05OUT DOCDATAKEY06OUT

Upon each iteration, these parameters will contain the application defined "keys" of a pending outbound transaction document that matched the criteria specified. If there are multiple sets of keys, only the first will be retrieved (see note below).

**Note:** It is possible for a single pending oubound transaction document to contain multiple individual messages, each of which may have their own set of staging database key values. This typically happens, for example, for a complex outbound EDI transaction. The output parameters containing the staging database keys will only contain the values for the FIRST message in this case. This consideration does not apply to documents that are registered using the TXDOC\_REGOUTBND, TXDOC\_REGOUTEDI or TXDOC\_REGOUTX12 activities, which permit only one message and therefore only one set of staging database keys to be registered.

# FOR\_EACH\_TXDOCT

This is an iterator activity. It will iterate for each transaction set (or message) registered for the specified transaction document. For each iteration, it provides the control numbers and transaction document register staging file keys that identify the transaction set (or message).

Typically a process using this activity specifies further processing directives nested beneath this iterator activity to perform further processing related to the transaction set (or message).

### **INPUT Parameters:**

### **DOCNUMBER** : Required

Specifies the transaction document envelope number whose registered transaction sets (messages) are to be iterated.

## **OUTPUT Parameters:**

## CTRLNUMBER\_IC :

Upon each iteration, this parameter will contain the interchange control number for the current transaction set (message).

## CTRLNUMBER\_GP:

Upon each iteration, this parameter will contain the group control number for the current transaction set (message).

## CTRLNUMBER\_MS :

Upon each iteration, this parameter will contain the the transaction set (message) control number for the current transaction set (message).

## **DOCNUMBER\_IC** :

Upon each iteration, this parameter will contain the the internal document interchange sequence number for the current transaction set (message). This value corresponds to the key field DXXINTID in the staging database files DXX2IN, DXX3GP and DXX4MS.

## **DOCNUMBER\_GP**:

Upon each iteration, this parameter will contain the the internal document group sequence number for the current transaction set (message). This value corresponds to the key field DXXGRPID in the staging database files DXX3GP and DXX4MS.

## **DOCNUMBER\_MS**:

Upon each iteration, this parameter will contain the the internal document transaction set (message) sequence number for the current transaction set (message). This value corresponds to the key field DXXMSGID in the staging database file DXX4MS.

### DOCDATAKEY01DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 :

Upon each iteration, these parameters will contain the "application-defined" staging file key values for the current transaction set (message).

# FOR\_EACH\_TXTLIN

This is an iterator activity. It will read each line (\*) from a text file and on each iteration output the text line. The processing logic nested beneath FOR\_EACH\_TXTLIN activity is repeated for each line read.

This activity is not intended for routine processing of large volumes of data. It can be useful however for transferring limited amounts of information between activities, transformation maps and the processing sequence variable pool.

(\*) Any of the line terminators (CR, CRLF, LF, NL or LFCR) will be used to indicate the end of a line.

### **INPUT Parameters:**

### TXTFILEPATH : Required

This parameter must contain the path of the text file to be read.

eg C:\memo.txt

or /memos/memo1.txt

### **OUTPUT Parameters:**

### TXTLINE

For each iteration, this output parameter will contain the value of the current text line read.

# FOR\_EACH\_VAR

This is an iterator activity. It will iterate for each processing sequence variable extant in the variable pool. The iteration does NOT include built-in variables or internal variables. On each iteration the output parameters provide the variable name, index and variable value. Because it is an iterator activity, you may nest processing sequence directives beneath the FOR\_EACH\_VAR activity that will be executed for each iteration.

This activity is mainly intended for diagnostic purposes. It would not normally be used in live, completed BPI solutions. As a diagnostic aid, it will log the variable name, index and value into the processing sequence log for each iteration. It may not be necessary, therefore, to include any processing sequence directives nested beneath this activity - the provided logging may be sufficient for your diagnostic purposes. NB: the Processing Sequence Editor will give a warning that the iterator item is empty, but this warning may be ignored.

The activity takes a "snapshot" of the state of the variable pool before beginning the iteration. The following applies to any changes that occur during the iteration as a result of executing other activities or processing sequence directives:

- processing sequence variables that are added will not be included in the iteration
- processing sequence variables that are removed or cleared will still be included in the iteration

However, changes to the VALUES of processing sequence variables WILL be reflected in the iteration if they occur BEFORE the affected variable is iterated.

### **INPUT Parameters:**

There are no input parameters.

# **OUTPUT Parameters:**

### VARNAME :

This output variable provides the name of the current processing sequence variable.

## VARINDEX :

This output variable provides the index of the current processing sequence variable. If the variable is not part of a list, the value will be 1 (one).

# VARVALUE :

This output variable provides the value of the processing sequence variable for the current index.

# FTP\_COMMANDLIST

This Activity connects to the FTP host specified in the FTP Configuration and then executes FTP commands from a command list file. The command list file may be specified on the FTP Configuration or overridden by the COMMANDLISTFILE parameter. The command list file is a text file.

An email Event notification named FTPCMDLSTFAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then an email will be sent. Refer to Event Maintenance for setting up Event notifications.

### **INPUT Parameters:**

## **FTPCONFIG** : Required

This parameter should contain the name of an FTP Configuration with the type of Command List. This Configuration can be created and maintained using the FTP Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

# **COMMANDLISTFILE** : Optional

If it contains a non-blank value it will be used for the command list. If this parameter is not provided, then the command list file from the FTP Configuration will be used. Refer to the 2.3.2 FTP Configuration for a description and examples of command formats.

## **OUTPUT Parameters:**

There are no output parameters.

## Processing

This Activity uses either the LANSA Integrator FTPService or SFTPService (depending on FTP configuration choices).

It uses the Connection information from the FTP Configuration to connect to the remote host.

The command list file is read, one line at a time. The line terminator may be a carriage return, line feed, new line, carriage return line feed or line feed carriage return. Lines with a first character of # are considered to be comments and are ignored. All other lines should contain a single FTP command which can be used by the LANSA Integrator FTPService or SFTPService. These commands are executed using the LANSA Integrator service and must conform to that format.

Any failure encountered when executing the commands will attempt to invoke the email Notification event, FTPCMDLSTFAILED. If this event is active then an email will be sent to the specified email address with a notification of the failure.

# FTP\_DIRLIST

This Activity retrieves a list of files from a remote host using FTP. It connects to the remote FTP host and retrieves the names of files from a directory on the remote host. Details of the remote host and directories is taken from an FTP Configuration. A list of the files retrieved is output. The list is output in two formats - one containing just the file names, the other containing the full local path and file name.

Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

An email Event notification named FTPDIRLISTFAIL is available in this Activity. If this event is active and a failure occurs in this Activity, then an email will be sent. Refer to Event Maintenance for setting up Event notifications.

### **INPUT Parameters:**

### FTPCONFIG : Required

This parameter should contain the name of an FTP Configuration. This Configuration can be created and maintained using the FTP Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

## **REMOTEDIRECTORY** : Optional

If it contains a non-blank value it will be used as the directory on the remote host from which the list of files are retrieved. If this parameter is not provided, then the remote host directory from the FTP Configuration will be used. Note that this is a required parameter if configuration of type command list is used.

## **GETLIKE** – Optional

If it contains a non-blank value it will be used as the pattern use to identify files to be listed. Note that if this is specified, only files matching the given pattern will be listed. If blank, values from the FTP Configuration will be used.

# **OUTPUT Parameters:**

### FILELIST

Specifies the name of a list variable that will contain a list of files from the remote host. The list will contain only the file name.

For example: order5.csv

# FILEPATH

Specifies the name of a list variable that will contain a list of files from the remote host. The list will contain the full path and file name. For example: /inftp/order5.csv

This list may be used as an alternative to the FILELIST

# Processing

This Activity uses either the LANSA Integrator FTPService or SFTPService (depending on FTP configuration choices).

It uses the Connection information from the FTP Configuration to connect to the remote host.

It changes directory on the remote host to the directory specified either in the input parameter or the FTP Configuration. It lists the files in that directory. If the parameter GETLIKE is specified or if the FTP Configuration is an Inbound configuration and contains a Get files like pattern, then only files which match that pattern are listed. FTP mode is changed to Binary if required.

The output lists will contain each file in the directory.

Any failure encountered when executing, will attempt to invoke the email Notification event, FTPDIRLISTFAIL. If this event is active then an email will be sent to the specified email address with a notification of the failure.

# FTP\_INBOUND

This Activity retrieves a list of files from a remote host using FTP. It connects to the remote FTP host and retrieves files from a directory on the remote host. The files are placed in a local directory. Details of the remote host and directories is taken from an FTP Configuration. A list of the files retrieved is output. The list is output in two formats - one containing just the file names, the other containing the full local path and file name.

Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

An email Event notification named FTPINFAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then an email will be sent. Refer to Event Maintenance for setting up Event notifications.

# **INPUT Parameters:**

## FTPCONFIG : Required

This parameter should contain the name of an FTP Configuration with type of Inbound. This Configuration can be created and maintained using the FTP Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

# **REMOTEDIRECTORY** : Optional

If it contains a non-blank value it will be used as the directory on the remote host from which files are retrieved. If this parameter is not provided, then the remote host directory from the FTP Configuration will be used.

# LOCALDIRECTORY - Optional

If it contains a non-blank value it will be used as the local directory into which the retrieved files are placed. If this parameter is not provided, then the local directory from the FTP Configuration will be used.

## **GETLIKE** – Optional

This parameter may specify a pattern to identify the files to be retrieved from the remote host.For example: (1) the pattern '\*.ord' will retrieve files with a file extension of 'ord', (2) 'ord\*' will retrieve files whose names start with the characters 'ord', irrespective of the file extension.

If specified, the value of this parameter overrides the specifications in the FTP inbound configuration for *Get like pattern*. If this parameter is not

specified, the values in the FTP inbound configuration for *Get selection* and *Get like pattern* will apply.

# **OUTPUT Parameters:**

# FILELIST

Specifies the name of a list variable that will contain a list of files retrieved from the remote host. The list will contain only the file name.

For example: order5.csv

# FILEPATH

Specifies the name of a list variable that will contain a list of files retrieved from the remote host. The list will contain the full path and file name.

For example: /inftp/order5.csv

This list may be used as an alternative to the FILELIST

# Processing

This Activity uses either the LANSA Integrator FTPService or SFTPService (depending on FTP configuration choices).

It uses the Connection information from the FTP Configuration to connect to the remote host.

It changes directory on the remote host to the directory specified either in the input parameter or the FTP Configuration. It lists the files in that directory. If the FTP Configuration contains a Get files like pattern or the GETLIKE parameter has been specified, then only files which match that pattern are listed. FTP mode is changed to Binary if required. Processing then cycles through the list of files to GET each from the remote directory to the local directory. If required it deletes from the remote directory each file successfully transferred. The FTP session is terminated by a QUIT command.

The output lists will contain each file retrieved.

Any failure encountered when executing, will attempt to invoke the email Notification event, FTPINFAILED. If this event is active then an email will be sent to the specified email address with a notification of the failure.

# FTP\_OUTBOUND

This Activity will transfer files from the local machine to a remote host by FTP. It connects to the remote FTP host and puts a copy of local files onto the remote host. The files to be transferred may be provided in a list or from a local directory. Details of the remote host and directories is taken from an FTP Configuration.

An email Event notification named FTPOUTFAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then an email will be sent. Refer to Event Maintenance for setting up Event notifications.

### **INPUT Parameters:**

## **FTPCONFIG** : Required

This parameter must contain the name of an FTP Configuration with type of Outbound. This Configuration can be created and maintained using the FTP Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

# FILELIST\_FTPOUT : Optional

If it contains a non-blank value it will be used to select the files to be FTP'ed to the remote host. If this parameter value is provided the

LOCALDIRECTORY parameter is not used.

The files contained in this list must be a full path and file name.

For example: Windows c:\mydir\file1.txt

IBM i /mydir/subdir/filex.xml

# LOCALDIRECTORY : Optional

This parameter is checked for only if the FILELIST\_FTPOUT parameter does not exist.

If LOCALDIRECTORY contains a non-blank value it will be used as the local directory from which files are copied. Otherwise, the local directory from the FTP Configuration will be used.

# **REMOTEDIRECTORY** : Optional

If it contains a non-blank value it will be used as the directory on the remote host into which files are put. If this parameter is not provided, then the remote host directory from the FTP Configuration will be used.

# **ARCHIVEPATHOK** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the archive directory path for successful transactions.

If this parameter is not provided, then the Local Archive Path (OK) directory from the FTP Configuration will be used.

## **ARCHIVEPATHER** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the archive directory path for transactions ending in error.

If this parameter is not provided, then the Local Archive Path (ER) directory from the FTP Configuration will be used.

### **OUTPUT Parameters:**

There are no output parameters.

### Processing

This Activity uses either the LANSA Integrator FTPService or SFTPService (depending on FTP configuration choices).

If no list of files is input, a list of files in the local directory is built. This list may be filtered by files with a particular extension as specified in the FTP Configuration.

The Connection information from the FTP Configuration is used to connect to the remote FTP server.

The processing changes directory on the remote host to the directory specified either in the input parameter or the FTP Configuration. It changes to Binary mode if required.

Processing then cycles through the list of files, either input or built from the local directory contents. Each file is PUT onto the remote host in the remote directory. The local file is moved to the appropriate archive directory as specified on the FTP Configuration. If no archive directories are specified, the local file is not moved.

The FTP session is ended with a QUIT command.

Any failure encountered when executing, will attempt to invoke the email Notification event, FTPOUTFAILED. If this event is active then an email will be sent to the specified email address with a notification of the failure.

# FTP\_SCRIPT

This activity will execute a script of FTP subcommands. It is supported only on IBM i servers.

The FTP script may contain substitution variables that are replaced at run-time by the values of Processing Sequence variables. See below for details on this feature.

Other supplied FTP activities (including FTP\_INBOUND, FTP\_OUTBOUND and FTP\_COMMANDLIST) use the LANSA Integrator FTPService in their implementation. However, in some cases it is desirable or necessary to take advantage of IBM i QSYS.LIB file system object support that is provided by the native IBM i5/OS FTP client but NOT available through the LANSA Integrator FTPService. This activity is provided for such cases. For example, this activity can be used for transferring IBM i5/OS save files (\*FILE objects with SAVF attribute) between IBM i systems.

Notes:

- 1. The script is executed using the native IBM i5/OS FTP client.
- 2. The FTP script will run to completion irrespective of any errors that may occur during its execution. If errors occur in the FTP operations, the activity does not end in error see the description of the output parameter FTPERRORCOUNT for more information.
- 3. Because the FTP functions performed by this activity are determined by the user-provided FTP script, it does not inherently have any "inbound" or "outbound" sense. The script can perform any combination of FTP subcommands supported by the i5/OS FTP client, including PUT/MPUT, GET/MGET or both.

### **INPUT Parameters:**

### FTPCONFIG : Optional

This parameter can contain the name of an FTP configuration. If specified, it is recommended that you use an FTP configuration of type 'Command List', although you can also specify an Inbound or Outbound configuration. This activity will, at most, use the following attributes of the FTP configuration:

- Remote Host
- Remote Port Address
- Remote User

- Remote Password

- Command list file

You may specify \*NONE for the 'Remote Host'. If you do so, then the 'Remote Port Address' will not be used and your script should usually contain an FTP 'open' subcommand to establish the connection to a remote host that you specify.

Similarly, if the FTP configuration is not specified, then your script must contain the FTP subcommands (such as 'open') necessary to establish the required connection along with the necessary credentials. In this case, you must provide the remote host, port (if necessary) and credentials through the FTP subcommands in the script.

# FTPSCRIPTFILE : Optional

This parameter is optional, but if it is not provided, then you must provide the FTP subcommand script either through the FTPSCRIPTLIST parameter or in the 'Command list file' field of the FTP configuration. If this parameter is provided, then it must specify the full path to a text file that contains the FTP subcommand script to be executed. See FTP Subcommand Script below for more information.

# FTPSCRIPTLIST : Optional

This parameter is optional, but if it is not provided, then you must provide the FTP subcommand script either through the FTPSCRIPTFILE parameter or in the 'Command list file' field of the FTP configuration. If this parameter is provided, then it must contain a list of the FTP subcommands to be executed. See FTP Subcommand Script below for more information.

### OUTPUT Parameters: FTPOUTPUT:

Upon completion, this parameter will contain a list of FTP log output lines generated by running the FTP script. (The FTP log results are also available in the Processing Sequence log, depending on the logging level in effect.)

# **FTPERRORCOUNT**:

Upon completion, this parameter will contain a count of the lines in the FTP log that the activity interprets as representing errors. This is defined as lines that begin with an FTP response code in the range 400-499 and 500-599.

Note that:

- the error count will be incremented for some "normal" messages such as '467 bytes transferred ...'

response codes counted as errors are generally issued by the FTP server
errors (usually syntax errors) issued by the FTP client may not have a recognisable response code and so may not be counted as errors

- in any event, it may be normal for some scripts to generate errors. For all these reasons, the activity does NOT return an error status if the apparent error count is greater than zero. It is up to you to test the output parameters to determine the success of the operation and to handle error conditions in your Processing Sequence.

# **FTP Subcommand Script**

Whether you specify your FTP script through the FTP configuration, or through the FTPSCRIPTFILE or FTPSCRIPTLIST parameters, you should compose your script according to the following guidelines:

- Your script can consist of any valid FTP subcommands supported by the FTP client software
- Each FTP subcommand should be on a separate line or in a separate list entry
- Lines or list entries beginning with '\*' are ignored and may contain comments
- If you have specified an FTP configuration and the 'Remote Host' is not \*NONE, then the activity will insert the remote user id and password from the FTP configuration in the first line of the script. In this case, there is no need to specify the credentials inside your script unless the remote FTP server requires additional account information or unless your script closes the connection and you subsequently wish to open another connection.

Refer to the documentation for the FTP client software for information on the FTP subcommands supported. On IBM i servers, you can start the FTP client by typing the command FTP RMTSYS(\*NONE) at a command line. When the FTP client has started type the command HELP and press Enter to access on-line help about the supported FTP subcommands.

### Substituting Processing Sequence Variables in the FTP Subcommand script

The activity supports substitution variables in the FTP script. When found, they will be replaced by the value of the named variable in the Processing Sequence variable pool.

You can specify a substitution variable in the following form:

```
%%var.<variable-name>%%
```

where <variable-name> is the name by which the variable is known in the Processing Sequence variable pool. See the example FTP script below for an example of using substitution variables in an FTP script.

If you specify substitution variables in your FTP script, it is your responsibility to ensure that the variables named exist and have valid values assigned to them in the Processing Sequence that uses the script. The variables values may be set by any supported means - for example, they might be received as Processing Sequence parameters, explicitly set with an ASSIGN directive, or set as the result of running some other activity. If the named variables do not exist in the Processing Sequence when the FTP script is executed, this activity will issue a warning but execution will continue (the variable reference is removed from the FTP script).

Note that the following advanced forms of variable references are NOT supported by this feature:

- 1. Compound or qualified variables for example: \*tradingpartner.xxx
- 2. Indexed variables for example: mylist(3)

Typically you could circumvent these limitations, if necessary, by assigning the desired compound or indexed variables to a simple variable name before executing the script.

## **Example FTP Subcommand Script**

The following script might be used with a correctly configured FTP configuration to copy the contents of a source library on the source IBM i system to a target library on the target IBM i system. The processing sequence must set the value of the processing sequence variables SOURCELIB and TARGETLIB in order to specify the respective library names. The FTP configuration used must specify the remote host name and the remote user and password used to connect to the remote host.

\*\*\*\*\*

\* This is a sample FTP script \*\*\*\*\*\*\*\*\*\*\*

BINARY NAMEFMT 1 SENDEPSV 0 CWD /QSYS.LIB/%%var.TARGETLIB%%.LIB/ MPUT /QSYS.LIB/%%var.SOURCELIB%%.LIB/ CLOSE

# QUIT

\*\*\*\*\*

# **GET\_DTAARA**

This activity reads a value from a specified data area. It is supported only on IBM i servers.

This activity can only read from data areas created with TYPE(\*CHAR) and the starting position and length specified must be valid for the definition of the data area specified. The data area must exist and the job running the activity must have the necessary authorities.

### **INPUT Parameters:**

### **DTAARA** : Required

This parameter specifies the name of the data area.

LIB : Optional

This parameter can specify the name of the library containing the data area. If it is not specified, the processing sequence will use the library list to locate the data area.

### **START** : Required

This parameter specifies the starting position in the data area from which to read the value.

**LENGTH** : Required

This parameter specifies the length of the value to be read.

# **OUTPUT Parameters:**

### **STRINGOUT**:

Upon successful completion, this parameter will contain the value read from the specified positions in the specified data area.

# HASH\_FILE

This activity generates a hash value of the contents of the specified file according to a known algorithm.

The hash value is a fixed-length (according to the chosen algorithm), nonreversible representation of the contents of the file. Such hash values have many uses in information security, including as a means of detecting (accidental or intentional) changes to or corruption of the source data.

For example, you could generate and store a hash value for file A. Then, in another place or time, you could generate a hash value (using the same algorithm) for a file that purports to be file A, and by comparing the hash values you can establish that the files are the same (with a very high degree of certainty, depending on the algorithm chosen), or not (with absolute certainty).

The HASH\_FILE activity returns the generated hash value in string form, either as a hexadecimal representation, or in base64-encoded form.

Note that it is NOT possible to derive the original file contents from the hash value – that is, it is non-reversible. You should not seek to use this activity as a means of implementing reversible data encryption.

## **INPUT Parameters:**

## FILEPATH : Required

This parameter should contain the full name and path of the file for which the hash value is to be generated.

## ALGORITHM : Optional

This parameter can specify the hashing algorithm to be used to generate the hash value. If it is not specified, a default of SHA is assumed. You can choose from the following supported algorithms:

MD2 The MD2 Message-Digest Algorithm is optimized for 8-bit computers and is no longer considered secure.

It generates a 128-bit digest (hash) value, 22 characters when \*BASE64 encoded, or 32 characters when \*HEX encoded.

MD5 The MD5 Message-Digest Algorithm has been employed in a wide variety of security applications, and is also commonly used to check data integrity.

It generates a 128-bit digest (hash value), 22 characters when \*BASE64 encoded, or 32 characters when \*HEX encoded.

SHA SHA stands for "secure hash algorithm", and the SHA parameter value here implements the SHA-1 algorithm from the SHA family of algorithms.

It generates a 160-bit digest (hash value), 27 characters when \*BASE64 encoded, or 40 characters when \*HEX encoded.

SHA256 SHA256, SHA384 and SHA512 are a set of algorithms from the

- SHA384 SHA-2 family.
- SHA512 They generate 256, 384 or 512-bit (respectively) digests (hash values), 43, 64 or 86 characters when \*BASE64 encoded, or 64, 96 or 128 characters when \*HEX encoded.

### **ENCODING** : Optional

This parameter specifies the form of encoding applied to the generated digest or hash value (which is binary data) to produce the string representation given in the HASH output parameter. If not specified, a default of \*BASE64 is assumed. You can choose from the following options:

- \*BASE64 The result is a base64-encoded representation of the generated hash value.
- \*HEX The result is a hexadecimal string representation of the generated hash value.

### OUTPUT Parameters: HASH:

Upon successful completion, this parameter will contain the string representation of the generated hash value of the contents of the specified file, encoded according to the value of the ENCODING parameter.

# HTTP\_GET

This activity may be used to retrieve a document from an HTTP server (usually using the HTTP GET method).

This activity is used with an HTTP outbound configuration which can specify most of the parameters necessary to establish the connection to the HTTP server. Selected attributes of the HTTP outbound configuration may be overridden by the parameters to this activity.

A notification event named HTTPFAILED is available for this activity. If this event is active and a failure occurs in the activity, then the notification event will be fired to send an email or other supported form of notification.

## **INPUT Parameters:**

# HTTPCONFIG : Required

This parameter must specify the name of an HTTP outbound configuration that specifies most of the parameters necessary to establish the connection to the HTTP server. Selected attributes of the HTTP outbound configuration may be overridden by the other parameters to this activity, as described below.

## **URI** : Optional

This parameter may specify an alternate value used to override the URI that is specified in the HTTP outbound configuration. If used, the value should specify the URI to be used to connect to the remote host specified in the HTTP outbound configuration. If this parameter is not specified, then the value will be taken from the HTTP configuration.

## **RETURNFILEDIR** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the directory on the local server in which return payload file is saved (the file containing the HTTP response data received from the HTTP server). If this parameter is not provided, then the value will be taken from the Content File Directory specified in the HTTP outbound configuration.

# **RETURNFILENAME** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the return payload file name (the file containing the HTTP response data received from the HTTP server).

# HTTPMETHOD : Optional (Advanced)

This parameter may specify the HTTP request method to be used for the

outbound HTTP request. The HTTP protocol standard defines several methods and user-defined methods can be used for REST style applications. The default value is GET and that would be the most usual value when you are using the HTTP\_GET activity to retrieve a document from an HTTP server. One common alternate HTTP method is POST - you can use the HTTP\_POST activity to issue POST requests - you would use POST when you need to send data to the HTTP server and optionally receive response data.

### HTTPHEADERNAME: Optional (Advanced) HTTPHEADERVALUE - Optional (Advanced)

These two parameters may optionally be used to specify one or more HTTP header names and corresponding HTTP header values to be added to the HTTP header for the outbound request. Note that the activity already adds several common and "standard" properties to the request header. You might use these parameters when your application requires you to specify additional HTTP header properties such as 'AUTHORIZATION' or custom properties specific to the HTTP server application. The parameters are lists, allowing you to specify multiple values in each list by assigning them using a list index. The activity will add as many properties as you specify in the HTTPHEADERNAME list. If used, it is your responsibility to ensure that each HTTPHEADERNAME list entry has a corresponding HTTP header property value.

### **OUTPUT Parameters:**

There are no output parameters.

# HTTP\_INBOUND

This activity may be used to process an inbound HTTP request. It will receive the content of the HTTP request and save it to a specific directory. The name and location of the saved content will be output as a parameter value.

Other identifying criteria relating to the HTTP message will be output as parameter values.

This activity is designed to be invoked in the following way:

- 1. LANSA Integrator's JSMDIRECT CGI program is invoked to handle the incoming request
- 2. JSMDIRECT uses the application name specified in the URI of the incoming request to determine the name of a program or LANSA function to process the request. In the context of the HTTP inbound support in LANSA Composer, this is usually the supplied DXHTFN1 function.
- 3. Function DXHTFN1 uses the application name specified in the URI of the in-coming request to determine the name of a LANSA Composer processing sequence to process the request.
- 4. The processing sequence should include this activity to receive and save the HTTP content.

In LANSA Composer, the application name specified in the URI of the incoming request is associated with the function (usually DXHTFN1) and the processing sequence by means of an HTTP inbound configuration. Creating or modifying an appropriate HTTP inbound configuration will also maintain the necessary entries in the tables used by JSMDIRECT.

This activity is nominally an iterator activity, although it is unusual insofar as it iterates only once. It is defined as an iterator activity in order to allow any processing sequence directives nested beneath it to process the received content and prepare the response payload, if any. You can nest transformation maps, other activities or whatever directives you need to perform this. The activity's input parameters (RESPONSEFILEPATH and RESPONSECONTENTTYPE) that specify the response payload are not evaluated until the iteration completes - so your nested processing sequence directives can set their values.

### **INPUT Parameters:**

### **RESPONSEFILEPATH** : Optional

This parameter can specify the path and file name of a file containing content

(of the type specified in the RESPONSECONTENTTYPE parameter) that is to be sent in response to the inbound HTTP request. If specified, the content of the file is sent in response. If not specified, then the activity sends a 200 (OK) response status along with the OKHTTP response message specified in the HTTP configuration (if any).

Note that this parameter is not evaluated until the iteration completes - so your nested processing sequence directives can set its value according to the nature or content of the request received.

# **RESPONSECONTENTTYPE** : Optional

If you specify the RESPONSEFILEPATH parameter, then you must specify the corresponding content type (eg \*TEXTPLAIN) for the response content. This parameter is not used if the RESPONSEFILEPATH parameter is not specified.

Note that this parameter is not evaluated until the iteration completes - so your nested processing sequence directives can set its value according to the nature or content of the request received.

# OUTPUT Parameters:

# HTTP\_CONTENT

This value will contain the path and file name of the saved inbound HTTP content.

For example : On an IBM i server /inorders/HI25.dat

On a Windows server C:\order\HI123.dat

The directory will be taken from the system setting *HTTP inbound payload directory*.

The file name will be derived from a prefix plus a unique number plus .dat extension. The prefix is taken from the system setting *HTTP inbound filename prefix*.

# HTTP\_CONTENT\_TYPE

This value will contain all the content types which are applicable to the content.

For example, a Comma Separated file will have a value of \*TEXT \*SV \*CSV

Refer to the 2.3.3 HTTP Configuration for a full list of types.

# HTTP\_CONTENT\_TYPE2

This is an alternate representation of the content type. It will contain the

actual value of the CONTENT-TYPE HTTP header. For example, text/plain. **HTTP\_CONTENT\_LENGTH** 

This value will contain the content length of the content received on the HTTP inbound request.

# HTTP\_QUERY\_STRING

This value will contain the query string from the URI.

If the full URL was http://lansa01:8080/cgi-bin/jsmdirect?ZZorders then the query string would have a value of ZZorders.

### HTTP\_REMOTE\_ADDRESS

This value will contain the IP address of the remote server.

### HTTP\_REMOTE\_USER

This value will contain the remote user if available.

### HTTP\_SERVER\_PORT

This value will contain the local port number that serviced this inbound HTTP message.

### Function DXHTFN1 - supplied JSMDirect function.

Function DXHTFN1 in process DXPROC02 is supplied to be invoked by HTTP JSMDirect.

This function will retrieve the query string from the URI. It will trim the query string to remove any additional parameters. The resulting string is used to locate the Composer HTTP Inbound Configuration with the corresponding Application Name.

The Processing Sequence on that Composer HTTP Configuration is then launched. This Processing Sequence should contain the HTTP\_INBOUND Activity to retrieve the payload and information from the inbound HTTP transmission.

# HTTP\_POST

This activity may be used to send data to an HTTP server (usually using the HTTP POST method) and/or to receive response data from the server.

This activity is used with an HTTP outbound configuration which can specify most of the parameters necessary to establish the connection to the HTTP server and to identify the file to be sent to the HTTP server. Selected attributes of the HTTP outbound configuration may be overridden by the parameters to this activity.

A notification event named HTTPFAILED is available for this activity. If this event is active and a failure occurs in the activity, then the notification event will be fired to send an email or other supported form of notification.

# **INPUT Parameters:**

# HTTPCONFIG : Required

This parameter must specify the name of an HTTP outbound configuration that specifies most of the parameters necessary to establish the connection to the HTTP server and to identify the file to be sent to the HTTP server. Selected attributes of the HTTP outbound configuration may be overridden by the other parameters to this activity, as described below.

## **URI** : Optional

This parameter may specify an alternate value used to override the URI that is specified in the HTTP outbound configuration. If used, the value should specify the URI to be used to connect to the remote host specified in the HTTP outbound configuration. If this parameter is not specified, then the value will be taken from the HTTP configuration.

# **CONTENTFILEDIR** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the directory on the local server in which the content payload file is found (the file containing the data to be sent to the HTTP server). If this parameter is not provided, then the value will be taken from the HTTP configuration.

## **CONTENTFILENAME** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the content payload file name (the file containing the data to be sent to the HTTP server). If this parameter is not provided, then the value will be taken from the HTTP configuration.

## **CONTENTFILETYPE** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the content type. If this parameter is not provided, then the value will be taken from the HTTP Configuration.

If the content file type in this parameter *and* in the HTTP configuration are blank, or specify the special value \*AUTO, then the content type will be derived from the file extension of the file being sent. (This is achieved by a lookup on the file *filetype.txt* in the *<system>* directory of the server's JSM instance directory tree.)

Refer to the 2.3.3 HTTP Configuration, *Outbound*, for the format and possible special "shorthand" values you may specify for this parameter, such as \*XML, \*HTML, \*CSV, \*PDF and \*ZIP. However, you are not limited to those values – you may explicitly specify an *actual* content type such as *application/zip*.

# **RETURNFILEDIR** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the directory on the local server in which return payload file is saved (the file containing the HTTP response data received from the HTTP server). If this parameter is not provided, then the value will be taken from the CONTENTFILEDIR.

## **RETURNFILENAME** : Optional

This parameter is optional. If it contains a non-blank value it will be used as the return payload file name (the file containing the HTTP response data received from the HTTP server).

# HTTPMETHOD : Optional (Advanced)

This parameter may specify the HTTP request method to be used for the outbound HTTP request. The HTTP protocol standard defines several methods and user-defined methods can be used for REST style applications. The default value is POST and that would be the most usual value when you are using the HTTP\_POST activity to "send" a file to a remote server via HTTP. One common alternate HTTP method is GET - you can use the HTTP\_GET activity to issue GET requests - you would use GET when you have no content to send but you are using the activity to receive content from the remote HTTP server.

## HTTPHEADERNAME: Optional (Advanced) HTTPHEADERVALUE - Optional (Advanced)

These two parameters may optionally be used to specify one or more HTTP

header names and corresponding HTTP header values to be added to the HTTP header for the outbound request. Note that the activity already adds several common and "standard" properties to the request header. You might use these parameters when your application requires you to specify additional HTTP header properties such as 'AUTHORIZATION' or custom properties specific to the HTTP server application. The parameters are lists, allowing you to specify multiple values in each list by assigning them using a list index. The activity will add as many properties as you specify in the HTTPHEADERNAME list. If used, it is your responsibility to ensure that each HTTPHEADERNAME list entry has a corresponding HTTPHEADERVALUE list entry containing the corresponding HTTP header property value.

#### **OUTPUT Parameters:**

There are no output parameters.
# JSM\_RECLAIM

This activity reclaims LANSA Integrator JSM resources by initiating garbage collection in the JSM's Java Virtual Machine instance.

In normal circumstances, garbage collection is entirely managed by the JVM and it is not necessary to intervene.

In some instances, however, the delay in garbage collection can lead to functional issues in subsequent processing when certain resources, such as files, might remain locked awaiting garbage collection of Java objects that reference them.

For example, if a Transformation Map ends with an exception, it can sometimes leave input or output files locked, preventing the processing sequence performing further operations on them.

In such cases, this activity, frequently executed conditionally by placing it in a CATCH block, may avoid the file locking issues.

Excessive use of this activity should be avoided as unnecessarily pre-empting normal garbage collection processing can adversely affect performance of your solution.

# **INPUT Parameters:**

# JSMCOUNT : Optional

This parameter specifies the number of iterations of garbage collection to initiate in the LANSA Integrator JSM's Java Virtual Machine instance. More than one may be necessary in some instances, as each iteration simply triggers one "round" of garbage collection - it is not necessarily exhaustive. The value specified must be numeric. If not specified, a default of 3 (three) is assumed.

# **OUTPUT Parameters:**

# JSM\_SCRIPT

This activity will execute a script of LANSA Integrator JSM commands.

The JSM command script may contain substitution variables that are replaced at run-time by the values of Processing Sequence variables. See below for details on this feature.

Note that JSM commands executed using this activity are not able to pass or receive LANSA variables or working lists to or from the LANSA Integrator service. This fact will restrict the services and/or the selection of service commands that may usefully be executed using this activity.

Due to this restriction and limited error handling and recovery capabilities, the activity is intended only for relatively simple LANSA Integrator service command sequences. You should consider using one of the purpose-specific supplied activities or creating your own custom activity for more advanced applications.

As soon as this activity begins to execute the JSM command script, the restartable flag is set OFF. It is not possible to restart a processing sequence that ends in error in this activity after it begins to execute the JSM command script.

## **INPUT Parameters:**

# JSMSERVICE : Optional

This parameter can contain the name of the JSM service to be loaded. If specified, the activity will load the JSM service at the beginning of execution and unload it when complete. If a single service is being used, this is the preferred technique because it allows the activity to respect and exploit applicable LANSA Composer settings and functionality, such as LANSA Composer features supporting LANSA Integrator tracing. If not specified, then the JSM script must contain the necessary SERVICE\_LOAD and SERVICE\_UNLOAD commands to load and unload the required JSM service.

# JSMSCRIPTFILE : Optional

This parameter is optional, but if it is not provided, then you must provide the JSM command script through the JSMSCRIPTLIST parameter. If this parameter is provided, then it must specify the full path to a text file that contains the JSM command script to be executed. See JSM command Script below for more information.

JSMSCRIPTLIST : Optional

This parameter is optional, but if it is not provided, then you must provide the JSM command script through the JSMSCRIPTFILE parameter. If this parameter is provided, then it must contain a list of the JSM commands to be executed. See JSM command Script below for more information.

# JSMERRORACTION : Optional

This parameter specifies the action the activity should take if any command in the specified JSM command script results in an error. You can specify one of the following values:

\*ERROR : the activity immediately ends in error without executing further JSM script commands

\*CONTINUE : the activity continues to attempt to execute remaining JSM script commands, but when complete, ends with a warning status

If not specified (or if an unrecognised value is specified), the activity assumes a default value of \*CONTINUE.

#### OUTPUT Parameters: JSMSTATUS:

Upon completion, this parameter will contain a list of the JSM status codes (eg: 'OK' 'ERROR' ...) resulting from executing the JSM script commands. Only status codes resulting from script commands are included - the list does NOT include status codes resulting from opening or closing the JSM connection or from loading or unloading the service (unless the service load or unload results from commands in the JSM script). Therefore the sequence of the status codes in the list corresponds directly to the sequence of the JSM commands executed.

# **JSMMESSAGE**:

Upon completion, this parameter will contain a list of the JSM messages resulting from executing the JSM script commands. Only messages resulting from script commands are included - the list does NOT include messages resulting from opening or closing the JSM connection or from loading or unloading the service (unless the service load or unload results from commands in the JSM script). Therefore the sequence of the JSM messages in the list corresponds directly to the sequence of the JSM commands executed.

# JSMERRORCOUNT:

Upon completion, this parameter will contain a count of the JSM command script commands that ended in error. If \*ERROR is specified for the

JSMERRORACTION parameter, this count will always be either zero or 1 because the first error terminates the JSM command script execution.

## **JSM Command Script**

Whether you specify your JSM command script through the JSMSCRIPTFILE or JSMSCRIPTLIST parameters, you should compose your script according to the following guidelines:

- Your script can consist of any JSM service commands valid for the JSM service loaded and according to the JSM service documentation, providing that the commands used do not rely upon passing program variables or lists to or from the LANSA Integrator service
- Each JSM service command should be on a separate line or in a separate list entry
- Lines or list entries beginning with '\*' are ignored and may contain comments

Refer to the LANSA Integrator documentation for information about the supported JSM services and service commands.

## Substituting Processing Sequence Variables in the JSM command script

The activity supports substitution variables in the JSM command script. When found, they will be replaced by the value of the named variable in the Processing Sequence variable pool.

You can specify a substitution variable in the following form:

```
%%var.<variable-name>%%
```

where <variable-name> is the name by which the variable is known in the Processing Sequence variable pool. See the example JSM command script below for an example of using substitution variables.

If you specify substitution variables in your JSM command script, it is your responsibility to ensure that the variables named exist and have valid values assigned to them in the Processing Sequence that uses the script. The variables values may be set by any supported means - for example, they might be received as Processing Sequence parameters, explicitly set with an ASSIGN directive, or set as the result of running some other activity. If the named variables do not exist in the Processing Sequence when the JSM command script is executed, this activity will issue a warning but execution will continue (the variable reference is removed from the JSM command).

Note that the following advanced forms of variable references are NOT

supported by this feature:

- 1. Compound or qualified variables for example: \*tradingpartner.xxx
- 2. Indexed variables for example: mylist(3)

Typically you could circumvent these limitations, if necessary, by assigning the desired compound or indexed variables to a simple variable name before executing the script.

# **Example JSM Command Script**

This is a sample JSM command script that is intended to delete records from the LANSA Composer tutorial orders database tables TUTORDH and TUTORDL using the LANSA Integrator SQLService. It uses substitution variables for the database user and password and for the BCHNUM database field value that identifies the records to be deleted. It is the responsibility of the solution designer to ensure that corresponding processing sequence variables exist with appropriate values before executing this script using the JSM\_SCRIPT activity.

Note that the example above would require entries similar to the following to be added to LANSA Integrator's SQLService.properties file in order to operate successfully. This is a requirement of the particular JSM service and service commands being used - it does not apply to the JSM\_SCRIPT activity in general. You should refer to the LANSA Integrator documentation for information about requirements and considerations that apply to the JSM service and service and service and service commands that you are using.

driver.tut=com.ibm.as400.access.AS400JDBCDriver
database.tut=jdbc:as400://SYSTEM/LICLICLIB;naming=sql;errors=full;date
format=iso;translate binary=true
#

# LAST\_SPLF

This activity retrieves the identity of the last spooled file created for the current job. This activity is only supported on IBM i servers.

You might use this activity after executing another activity that you expect will have created a spooled file in the current job. For example, executing certain system commands (using the SYSTEM\_COMMAND activity), a custom activity or calling a program or LANSA function may create a spooled file. Note that this activity will ONLY retrieve the identity of a spooled file created in the same job that executes the LAST\_SPLF activity. This means that there are several common circumstances in which this activity will NOT yield the desired information. For example:

- spooled files generated as a consequence of executing a LANSA Integrator service command
- calling a LANSA function, if the request is satisfied through the LANSA Composer Request Server

(You may use the SPLF\_LIST activity to list spooled files created in another job or by another user.)

You should check that the spooled file is the one you expect before processing it further - for example, you might at least test that the SPLFNAME value is the spooled file name that you expect.

Note that in some circumstances (in particular when the current job's user name is not the same as the current user profile), the job details for the spooled file will not match the job name, user name and job number for the current job. For example, you may find that a job named QPRTJOB owns the spooled file. You should use the details provided by this activity or by the SPLF\_LIST activity if you wish to perform further operations on the spooled file.

If no last spooled file information is available, the output parameters will be empty. The activity ends normally (without error) in this case.

## **INPUT Parameters:**

There are no input parameters for this activity.

### OUTPUT Parameters: JOBNAME:

If successful, this parameter will contain the name of the job that owns the spool file.

#### **JOBUSER** :

If successful, this parameter will contain the user profile name of the job that owns the spool file.

### **JOBNUMBER**:

If successful, this parameter will contain the job number of the job that owns the spool file.

#### **SPLFNAME :**

If successful, this parameter will contain the name of the spool file.

### **SPLFNUMBER**:

If successful, this parameter will contain the number of the spool file.

# LOAD\_PSVSET

This activity will load processing sequence variables from one or more PSV sets (processing sequence variables sets) contained in the specified PSV file.

Because this is an iterator activity, the processing logic nested beneath the LOAD\_PSVSET activity is repeated for each PSV set loaded. In some valid usage scenarios for this activity, the process may only expect to load one PSV set - for example when loading a PSV file that was created using the SAVE\_PSVSET activity. In that instance it is the process designer's choice whether or not to nest the associated processing directives beneath this activity.

There is no inherent limit to the number of PSV sets that can be loaded or the number of processing sequence variables loaded from each PSV set. However, the activity is not intended for routine processing of large volumes of data and performance may suffer if you attempt to do so.

The PSV files processed by this activity may contain one or more PSV sets (<psvSet> element). PSV files created by the SAVE\_PSVSET activity will always contain a single PSV set (<psvSet> element). However, PSV files created by other means (for example, as the output from a Transformation Map) may contain more than one PSV set (processing sequence variables set).

For more information about PSV files, refer to the following.

Saving, Loading and Transforming Processing Sequence Variables

NOTE: This activity does not clear any processing sequence variables or variable lists before loading them from the PSV file. This is particularly important for variable lists. If your processing sequence already contains a list named &my\_list containing 30 entries, and a list of the same name is loaded from the PSV file with 15 entries, the resulting list will still contain 30 entries - the LOAD\_PSVSET activity will replace the values of only the first fifteen entries in this instance. You will need to use the CLEARLIST activity if you wish to ensure that any loaded variable lists contain ONLY the entries loaded from the PSV file. If you are using the iterator capability of this activity (to load multiple PSV sets), then the CLEARLIST activity would typically need to be repeated at the end of each iteration.

**INPUT Parameters: PSVFILEPATH** : Required This parameter must specify the path and file name of the PSV file to be read.

# **PSVSELECT** : Optional

If the PSV file read by this activity contains more than one PSV set (<psvSet> element), this parameter may be used to select a single PSV set (processing sequence variables set) to be loaded. To do this, you may specify the special value '\*FIRST' to select the first processing sequence variables set in the PSV file or you may specify an id value to be matched against the value of the id= attribute of the <psvSet> element. If you specify an id, the match is case-sensitive. If this parameter is not specified or the default value of '\*ALL' is used, then the activity will iterate for each processing sequence variables set present in the PSV file.

#### **OUTPUT Parameters: PSVINDEX**:

For each iteration, this output parameter will contain the index of the current PSV set in the PSV file. If you specify an id value for the PSVSELECT parameter, the returned index may not necessarily be 1 (one) if the selected PSV set was not the first in the file.

# **PSVSETID**:

For each iteration, this output parameter will contain the value of the id= attribute of the <psvSet> element for the current PSV set.

# **PSVCOMMENT**:

For each iteration, this output parameter will contain the value of the comment= attribute of the psvSet> element for the current PSV set.

# LOGLIST

This activity creates one or more entries in the processing sequence log that contain the values of the variable list specified in the parameter. This can be useful while testing or debugging your processing sequence.

Note that logging is subject to the logging level in effect for the processing sequence run.

#### **INPUT Parameters:**

#### **LIST** : Required

Specifies the list values to be logged. Normally you would specify a variable list in this parameter - if you do so, LANSA Composer will log the indexed variable name and its value for each item in the list.

(It is possible, but not useful, to specify a literal value for this parameter.)

## **OUTPUT Parameters:**

# LOGUSERINFO

This activity creates an impromptu entry in the processing sequence log - subject to the logging level in effect for the processing sequence run.

## **INPUT Parameters:**

## **LOGTEXT** : Required

Specifies the impromptu message text for log entry. The activity will use a maximum of 512 characters, but, in any event, you should usually specify considerably less than this.

## LOGEXTRA : Required

Specifies additional information that supports the impromptu message. This might enumerate parameter values or a command string that affect the operation in progress. This information, if specified, can be displayed for the log entry under the heading additional information. The activity will use a maximum of 512 characters for this parameter.

#### **OUTPUT Parameters:**

# LOGVARIABLE

This activity creates an entry in the processing sequence log that contains the value of the variable specified in the parameter. This can be useful while testing or debugging your processing sequence.

Note that logging is subject to the logging level in effect for the processing sequence run.

### **INPUT Parameters:**

#### VARIABLE : Required

Specifies the value to be logged. Normally you would specify a variable in this parameter - if you do so, LANSA Composer will log the variable name and its value.

(It is possible, but not useful, to specify a literal value for this parameter.)

## **OUTPUT Parameters:**

# LOWERCASE

This activity returns the specified input string with all uppercase characters converted to lowercase.

## **INPUT Parameters:**

**STRINGIN** : Required

This parameter specifies the string to be converted.

#### **OUTPUT Parameters: STRINGOUT** :

Upon completion, this parameter will contain the converted string.

# MAIL\_RECEIVE

This Activity will retrieve a single email message from the mail server using LANSA Integrator's POP3MailService. It will retrieve the first email message for a particular user. The user details and other criteria must be specified in a POP3 Mail Configuration.

Information about the email, the mail text and attachments will be output.

If the "Remove from server after read" is set to Yes on the POP3 Mail Configuration, then the next execution of this Activity will retrieve the next email message.

An email Event notification named POP3FAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then a notification email will be sent. Refer to Event Maintenance for setting up Event notifications.

## **INPUT Parameters:**

## **POP3CONFIG** : Required

This parameter must contain the name of a POP3 Mail Configuration. This Configuration can be created and maintained using the POP3 Mail Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

# **SAVEDIRECTORY** : Optional

This parameter may be used if the POP3 Configuration requires mail text or attachments to be saved. If it contains a non-blank value it will be used as the directory into which the email message text and attachments will be saved.

If this parameter is not provided, and text or attachments are to be saved, then the directory from the POP3 Configuration will be used.

## **OUTPUT Parameters:**

## POP3COUNT

Upon completion, this value will contain a count of the email messages received. Because this activity receives a maximum of one message, this value will be either zero or one.

## FROMADDRESS

This value will contain the from address from the retrieved email message. **SUBJECT** 

This value will contain the Subject from the received email message.

#### SENTDATE

This value will contain the date on which the received email message was sent.

#### ATTACHMENTNO

This will contain the number of attachments on the received email message.

#### TEXTFILE

If the POP3 Configuration requires the email text to be saved, then this value will contain the path and file name of the email text which has been saved.

For example: Windows c:\messages\mailtext.txt

IBM i /email/mailtext.txt

### ATTACHMENTLIST

If the POP3 Configuration requires the email attachments to be saved, this will contain a list of full path and file name which have been saved from the received email.

The attachments will be saved in the directory specified in the POP3 Mail Configuration or the input parameter SAVEDIRECTORY.

# MAIL\_RECEIVEALL

This Activity will retrieve multiple email messages for a user from the mail server using LANSA Integrator's POP3MailService.

Information about the email, the mail text and attachments from the retrieved emails will be saved to a local directory. A directory will be created for each email retrieved. A list of the directories created will be output by this Activity.

The parent directory for these created directories may be specified in the POP3 Mail Configuration or in the input parameter SAVEDIRECTORY.

Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

An email Event notification named POP3FAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then a notification email will be sent. Refer to Event Maintenance for setting up Event notifications.

## **INPUT Parameters:**

# **POP3CONFIG** : Required

This parameter must contain the name of a POP3 Mail Configuration. This Configuration can be created and maintained using the POP3 Mail Configuration option in the *Navigator*.

If this parameter is not provided or is not found, this Activity will return an error and processing will be abandoned.

# **SAVEDIRECTORY** : Optional

If this parameter contains a non-blank value it will be used as the home directory into which the sub directory to contain the email details is created. The details of the email message, the message text and attachments will be saved into this sub directory. If this parameter is not provided, then the directory from the POP3 Configuration will be used.

# **POP3MAX** : Optional

If this parameter contains a number, it will be used as the maximum number of emails that this activity will retrieve. A special value of \*NOMAX can be used to indicate that there be no maximum. In this case, all emails that have not been deleted will be retrieved. If this parameter is not provided, then a default limit of 50 will be used.

# **OUTPUT Parameters:**

## POP3COUNT

Upon completion, this value will contain a count of the email messages received. The value may be zero if no mail was available to receive.

## STOREDIRECTORY

Specifies the name of a list variable that will contain a list of sub directories which have been created within the SAVEDIRECTORY. The full path will be held.

For example: Windows \savedir\subdir

IBM i /savedir/subdir

The name of the sub-directory will be generated from the system setting *POP3 save sub-directory name prefix* plus a unique number. For example : MS123

This sub-directory will contain:

- Mail attachments (if required to be saved)
- Mail text (if required to be saved) as file named mailtext.txt
- Mail information in file named mailinfo.txt Format of this mailinfo.txt is:

# MAIL\_SEND

This Activity will send an email with or without attachments to one or more recipients.

The details for the email must be specified in an SMTP mail details configuration. Selected details of the configuration may be overridden using the activity parameters.

To use this Activity you must have an SMTP mail server available. The activity will send the email through the SMTP server identified in the specified or system default SMTP server configuration.

An email Event notification named SMTPFAILED is available in this Activity. If this event is active and a failure occurs in this Activity, then an email will be sent. Refer to Event Maintenance for setting up Event notifications.

# **INPUT Parameters:**

# SMTPMESSAGEDETAILID : Required

This parameter should contain the name of an SMTP Message Detail configuration that specifies details of the e-mail message to be sent. If this parameter is not provided or is not found, an error will be returned and processing abandoned.

# SMTPSERVERID : Optional

This parameter may contain the name of SMTP server Configuration that will be used to send the e-mail. If this parameter is not provided then the default SMTP server configuration set in System Settings will be used as the mail server.

## **TOADDRESS** : Optional

This parameter may contain the email address(es) to be used for the TO email address. If not specified, then the TO email address(es) from the detail configuration will be used as the TO email address. You may specify a single email address (eg: 'me@here.com') or a list of email addresses separated by commas (eg: 'me@here.com,you@there.com').

# **CCADDRESS** : Optional

This parameter may contain the email address(es) to be used for the CC (copy to) email address. If not specified, then the CC email address from the detail configuration will be used as the CC email address. You may specify a single email address (eg: 'me@here.com') or a list of email addresses separated by commas (eg: 'me@here.com,you@there.com').

# BCCADDRESS : Optional

This parameter may contain the email address(es) to be used for the BCC (blind copy to) email address. If not specified, then the BCC email address from the detail configuration will be used as the BCC email address. You may specify a single email address (eg: 'me@here.com') or a list of email addresses separated by commas (eg: 'me@here.com,you@there.com').

# FROMADDRESS : Optional

This parameter may contain an email address to be used for the FROM email address. If not specified, then the FROM email address from the detail configuration will be used as the FROM email address.

NB: SMTP mail servers may enforce certain rules about the FROM addresses that can be specified, in order to prevent abusive practices such as mail relay.

# FROMDISPLAYNAME : Optional

This parameter may contain the FROM display name. If not specified, then the FROM display name from the detail configuration will be used as the FROM display name.

# **SMTPSUBJECT** : Optional

This parameter may contain the email subject line. If not specified, then the subject from the configuration will be used as the email subject line.

# MAILBODYTEXT:Optional

This parameter may contain the location and file name to be used as the body text.

If parameter values are available, it will be used for the body text of the email message.

If no parameters exist and a value exists on the detail Configuration, it will be used for the body text.

# **MAILATTACHMENT** : Optional

Mail attachments may be provided as a list.

If parameter values are available, they will be added as attachments.

If no parameters exist and a value exists on the detail Configuration, it will be used to add an attachment.

# MAILATTACHMENTZIP : Optional

If some mail attachments were found, they may be attached as a zip. The name of the zip will be retrieved from the parameters. If a parameter does not exist but a value for the zip exists in the details Configuration that name

will be used.

If no zip name is provided, as parameter or in Configuration, then attachments are added unzipped.

# **OUTPUT Parameters:**

# **MOVE\_FILE**

This Activity will move a file from one directory to another.

When executed on a Windows platform a system move command is executed.

When executed on an IBM i platform an i5/OS MOV command is executed. Note that files in the integrated file system are moved – if you want to move objects in the QSYS file system, you must use IFS file system notation to do so.

After the move the file will not exist in the original directory.

When executed on an IBM i platform, this activity is capable of performing data conversion between different CCSIDs during the move operation. For example, to convert the data in a text file from its original encoding to UTF-8 you could use the parameter values TOCCSID(1208) DTAFMT(\*TEXT).

# **INPUT Parameters:**

# FROMFULLNAME : Required

This parameter should contain the full path and name of the file to be moved

For example: Windows C:\dir1\myfile.txt

IBM i /indir/myfile.txt

# **TODIRECTORY** : Optional (\*)

This parameter may contain the full directory path of the destination directory. If specified, the file will be moved to this new location using the same name and extension as the original file. If not specified, you must provide a value for the TOFULLNAME parameter.**TOFULLNAME** : Optional (\*)

This parameter may specify the path of the directory the object is to be moved to AND the new name of the object. If specified, the file will be moved to this new location using the new file name and extension specified. If not specified, you must provide a value for the TODIRECTORY parameter.

# FROMCCSID : Optional, IBM i servers only

This parameter may specify the method for obtaining the coded character set identifier (CCSID) for the source of the move operation. This CCSID will be used for data conversion, if requested.

This parameter corresponds directly to the FROMCCSID parameter of the IBM i MOV command and you may use any values that are defined for the MOV command. At IBM i V7R1 the possible values include \*OBJ,

\*PCASCII, \*JOBCCSID or a CCSID value in the range 1-65533. Refer to the IBM documentation for the MOV command for more information.

The special (default) value \*DEFAULT specifies that the FROMCCSID parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

TOCCSID : Optional, IBM i servers only

This parameter may specify the data coded character set identifier (CCSID) for the target of the move operation.

This parameter corresponds directly to the TOCCSID parameter of the IBM i MOV command and you may use any values that are defined for the MOV command. At IBM i V7R1 the possible values include \*OBJ, \*CALC, \*PCASCII, \*STDASCII, \*JOBCCSID or a CCSID value in the range 1-65533. Refer to the IBM documentation for the MOV command for more information.

The special (default) value \*DEFAULT specifies that the TOCCSID parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

**DTAFMT** : Optional, IBM i servers only

This parameter may specify the format of the data in the file to be moved..

This parameter corresponds directly to the DTAFMT parameter of the IBM i MOV command and you may use any values that are defined for the MOV command. At IBM i V7R1 the possible values are \*TEXT and \*BINARY. Refer to the IBM documentation for the MOV command for more information.

The special (default) value \*DEFAULT specifies that the DTAFMT parameter will not be specified. Note that this parameter applies only when LANSA Composer is running on an IBM i server.

### **OUTPUT Parameters: FULLNAMEOUT**

Upon successful completion, this parameter will contain the full path and file name of the resulting destination file.

# MOVE\_SPLF

This activity will move a spool file on an IBM i server to a specified output queue. It is only supported on IBM i servers.

The input parameters must fully identify the spool file to be moved. You must specify the SPLFNUMBER parameter if there is more than one spool file of the name specified by the SPLFNAME parameter for the job specified by the first three parameters

### **INPUT Parameters:**

#### **JOBNAME** : Required

This parameter specifies the name of the job that created the spool file to be moved.

## JOBUSER : Required

This parameter specifies the user profile name of the job that created the spool file to be moved.

## JOBNUMBER : Required

This parameter specifies the job number of the job that created the spool file to be moved.

## **SPLFNAME** : Required

This parameter specifies the name of the spool file to be moved.

## SPLFNUMBER : Optional

This parameter specifies the number of the spool file to be moved and is only required if there is more than one spool file of the name specified by the SPLFNAME parameter for the job specified by the first three parameters. If not specified, a default special value of \*ONLY is used. You may specify special values \*ONLY or \*LAST. Otherwise specify the number of the job's spooled file that is to be moved.

## TOOUTQ : Required

Specifies the name of the destination output queue the spool file is to be moved to.

## TOOUTQLIB : Optional

Specifies the name of the library containing the destination output queue. The default value is \*LIBL meaning the library list is used to locate the named destination output queue.

## **OUTPUT Parameters:**

# MSG\_RECEIVE

This Activity will receive a message from a supported message brokering system such as IBM MQ Series. Refer to 2.3.9 Messaging Configuration for further information on this Activity and the pre-requisites for using it.

The MSGCONFIG parameter specifies the name of a messaging configuration that identifies, amongst other things, the address of and credentials for the messaging system and queue from which the message to be received.

The message contents are received into a file. The location and name of the received file is determined by the configuration in conjunction with the parameter values.

In addition, message properties can be received in comma-separated form into a second file. The location and name of the CSV file is determined by the configuration in conjunction with the parametervalues.

In all cases, parameter values, if specified, override values in the configuration.

When no message is available to be received (after waiting for the interval specified by the WAITTIME parameter, if used), the activity ends normally but the output parameters containing message attributes are not filled. The value of the MSGRCVCOUNT output parameter may be used to determine whether a message was received.

# **INPUT Parameters:**

# **MSGCONFIG** : Required

This parameter must contain the name of a messaging configuration. This configuration can be created and maintained using the Messaging Configurations option in the LANSA Composer client software. Amongst other things, the messaging configuration specifies the address, queue name and security credentials to be used with the message broker system. If this parameter is not provided or is not foundor is Inactive, this Activity will return an error and processing will be abandoned

# **WAITTIME**: Optional

Specifies the length of time (in milliseconds) that the activity will wait for a qualifying message to arrive in the message queue if one is not immediately available when the activity is processed.

The default value is -1 which means that the activity does not wait for the arrival of a message if one is not immediately available when the activity is processed.

You may also specify a special value of 0 (zero) which means that the activity waits indefinitely for a message to arrive. You should understand that this value may cause the activity and hence the processing sequence in which it is contained to stall indefinitely if no message is sent to the message queue. Other than by the arrival of a message, such a processing sequence run may only be terminated by terminating the job, the subsystem in which the job is running or the system.

# **ENCODING**: Optional

Use the ENCODING when the received message is a Unicode TextMessage. The default value is UTF-8.

# **MSGFOLDER**: Optional

Use the MSGFOLDER to specify the folder to which the file containing the received message contents will be written. If specified, this parameter overrides the Receive folder value in the messaging configuration.

You can leave this parameter unspecified and specify a fully qualified MSGFILENAME.

When neither the folder nor the MSGFILENAME are specified, the file will be created in \x\_lansa.

## MSGFILENAME: Optional

The name to be given to the file containing the received message contents. Use in conjunction with MSGFOLDER or the Receive folder in the configuration. If specified, this parameter overrides the Receive file name value in the messaging configuration.

When a filename is not specified, the file's name will be auto generated in the format:

<system setting prefix><number>

# **PTYFILENAME:** Optional

The name to be given to the file containing the received message properties. When this parameter has no value but message properties are received, the file will have the same name as the MSGFILENAME with the extension CSV appended to it.

The file will be created in the Message Properties Receive file path, if specified in the messaging configuration (Message Properties tab). If not, it will be created in the MSGFOLDER. Otherwise it will be created in \x\_lansa.

## **SELECTOR**: Optional

A message selector to be passed to the MQSeries system. Only valid for MQSeries message queue on an IBM i server. The message selector is used to select which message is to be received by the activity. If specified, only messages matching the specified criteria will be received by the activity. Otherwise, messages will not be filtered.

# OUTPUT Parameters:

# **MESSAGEID**:

The message ID of the received message as returned by the message broker system.

## **MSGRCVCOUNT**:

The received message count. If a message is received successfully, the value will be one (1). Otherwise the value will be zero (0).

## **MSGFILENAMEOUT:**

The full path and file name of the file to which the received message contents were written.

## **PTYFILENAMEOUT:**

The full path and file name of the file to which the received message properties were written.

# MSG\_SEND

This Activity will send a file through a supported message brokering system such as IBM MQ Series. Refer to 2.3.9 Messaging Configuration for further information on this Activity and the pre-requisites for using it.

The MSGCONFIG parameter specifies the name of a messaging configuration that identifies, amongst other things, the address of and credentials for the messaging system and queue to which the message is to be sent.

There are different ways to specify what file or files to send. Which way to choose depends entirely on your requirements.

Send a file located in a folder	Unqualified File name	Fully qualified folder name	N/A
Send a file located in a folder	Fully qualified file name	Blank	N/A
Send all files in a folder	Blank	Fully qualified folder name	Use default *.*
Send all files in a folder with extension gif	Blank	Fully qualified folder name	Gif

### MSGFILENAME MSGFOLDER MSGFILEEXTENSION

**Note:** the MSGFOLDER and MSGFILENAME parameters in this activity override the Send folder and Send file name in the messaging configuration. If they rarely change you can specify them at the configuration level and leave the MSGFOLDER and MSGFILENAME parameters blank.

## **INPUT Parameters:**

MSGCONFIG : Required

This parameter must contain the name of a messaging configuration. This configuration can be created and maintained using the Messaging Configurations option in the LANSA Composer client software. Amongst other things, the messaging configuration specifies the address, queue name and security credentials to be used with the message broker system. If this parameter is not provided or is not found or is Inactive, this Activity will return an error and processing will be abandoned.

# **MSGTYPE** : Optional

Set the MSGTYPE to \*BINARY to send the file contents as a binary bytes. Set the MSGTYPE to \*TEXT to send the file contents as a text message.The default value is \*BINARY.

# **ENCODING**: Optional

Used when the MSGTYPE is set to \*TEXT, the value of this parameter is used to convert the file content into a Unicode string that will be sent as the text message. The default value is UTF-8.

# **MSGFOLDER:** Optional

Use in conjunction with the MSGFILENAME or MSGFILEEXTENSION parameters. It must contain the fully qualified path to the folder containing the file or files to send.

Use the appropriate format depending on the operating system. For example:

```
Windows <drive>:\mydir\
```

# IBM i /mydir/subdir/

# **MSGFILENAME:** Optional

Use in conjunction with the MSGFOLDER parameter or the Send folder specified in the messaging configuration.

When specifying the folder, MSGFILENAME must contain the name of a file found in that folder.

When the folder is not specified, MSGFILENAME must contain a fully qualified file name.

Use the appropriate format depending on the operating system. For example:

Windows <drive>:\mydir\file1.txt

IBM i /mydir/subdir/filex.xml

# **MSGFILEEXTENSION**: Optional

This parameter can specify a file extension filter used to select files to be

sent from the folder specified in the MSGFOLDER parameter or in the messaging configuration. The filtering is done without regard to the case of the filter value or file extension. For example, you might specify a value of 'XML' for this parameter to select all files with an XML file extension. This parameter is ignored if a file name is specified in the MSGFILENAME parameter.

# **PTYFILENAME:** Optional

This parameter may specify the name of a file containing message properties that are to be merged with any specified in the messaging configuration and sent with the message(s).

If the Message Properties Send file path is specified in the messaging configuration, this parameter (if used) must specify the name of a file found in that folder.

If the Message Properties Send file path is NOT specified in the messaging configuration, this parameter (if used) must have the fully qualified file name.

Refer to 2.3.9 Messaging Configuration for information on the expected format of the message properties comma-separated file.

# OUTPUT Parameters:

## **MESSAGEID**:

The message IDs of the sent message(s) returned by the message broker system.

If the parameter values used result in more than one file/message being sent, then this outbound parameter contains a list of the message ids for each message sent.

# MSGQ\_RECEIVE

This activity receives a message from a named message queue on the IBM i server. It is supported only on IBM i servers.

When the activity receives a message from the named message queue, a range of attributes of the message are available through the output parameters of the activity, including message text, message data, message identifier and sender information.

When no message is available to be received (after waiting for the interval specified by the MSGWAIT parameter, if used), the activity ends normally but the output parameters containing message attributes are not filled. The value of the RCVCOUNT output parameter may be used to determine whether a message was received.

Refer also to the companion MSGQ\_SEND and WATCH\_MSGQ activities that allow you to send a message to a named message queue or to iteratively wait on messages arriving at a message queue. Refer to IBM i documentation concerning message queues, message descriptions and message files for further information about the operating system features upon which this activity depends.

# **INPUT Parameters:**

## MSGQ : Required

This parameter must specify the name of the message queue from which the message is to be received.

# MSGQLIB: Optional

This parameter may specify the name of the library containing the message queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named message queue.

# MSGPOS: Optional

You may specify this parameter to control which message is to be received from the message queue or to receive messages that are "old" (that is, messages that have previously been received without being removed).

The usual (and default) value, \*NEW, specifies that the activity will receive a "new" message, that is one that has not been previously received (without being removed). When this value is specified (or assumed) the messages are received in FIFO (first-in, first-out) order.

Alternatively you may use the values \*FIRST or \*LAST to specify that the

first or last message on the message queue is to be received. These values permit "old" messages to be received again, but will also receive "new" messages where applicable.

# MSGWAIT: Optional

Specifies the length of time (in seconds) that the activity will wait for a message to arrive in the message queue if one is not immediately available when the activity is processed.

The default value is 0 (zero) which means that the activity does not wait for the arrival of a message if one is not immediately available when the activity is processed.

You may also specify the special value \*MAX which means that the activity waits indefinitely for a message to arrive. You should understand that this value may cause the activity and hence the processing sequence in which it is contained to stall indefinitely if no message is sent to the message queue. Other than by the arrival of a message, such a processing sequence run may only be terminated by terminating the job, the subsystem in which the job is running or the system. If you wish a process to wait indefinitely for a message to arrive, the WATCH\_MSGQ activity may provide a more suitable solution.

# **MSGRMV**: Optional

Specifies whether the message received by the activity is removed from the message queue. The default is \*YES, which means the message is removed. You may specify \*NO to cause the message not to be removed from the message queue. If you do so, the system considers the message to be "old" and it will not be received again by this activity except by specifying \*FIRST or \*LAST for the MSGPOS parameter. If your processing sequence leaves "old" messages on the message queue, it is your responsibility to remove them by other means such as by executing the CLRMSGQ operating system command.

#### OUTPUT Parameters: RCVCOUNT:

The received message count. If a message is received successfully, the value will be one (1). Otherwise the value will be zero (0). The activity ends normally when no message is available to receive after waiting for the interval specified by the MSGWAIT parameter.

# **RCVMSGSEV**:

If a message is received, this parameter contains the severity code of the received message. Impromptu messages sent by the MSGQ\_SEND activity will usually have a severity code of '00'. For pre-defined messages, the severity code is defined by the message description.

# **RCVMSGTYPE**:

If a message is received, this parameter contains the message type of the received message. Messages sent by the MSGQ\_SEND activity will have a message type of \*INFO. For messages sent by other means, the message type is specified by the sending program.

# **RCVMSGID**:

If a message is received, this parameter contains the message identifier of the received message. An impromptu message has no message identifier. Otherwise, this value identifies the message description in the message file identified by the RCVMSGF and RCVMSGFLIB parameters.

# RCVMSGF:

If a message is received, this parameter contains the name of the message file of the received message. An impromptu message has no message file name. Otherwise, this value identifies the message file name containing the message description used to send the message.

# **RCVMSGFLIB**:

If a message is received, this parameter contains the name of the library containing the message file, if any.

# **RCVMSGTXT**:

If a message is received, this parameter contains the first level message text of the received message. For a pre-defined message, this includes the message data fields that were substituted for substitution variables in the text before the message was sent. A maximum of 256 characters of the first level text will be returned.

# **RCVSECLVL**:

If a message is received, this parameter contains the second level message text of the received message, if any. An impromptu message has no second level message text. For a pre-defined message, this includes the message data fields that were substituted for substitution variables in the second level text before the message was sent. A maximum of 256 characters of the second level text will be returned.

# **RCVMSGDTA**:

If a message is received, this parameter contains the message data field values of the received msg. An impromptu message has no message data. Otherwise this contains a single string of the message data values used to send the message. A maximum of 256 characters of message data will be returned.

# **SENDJOB**:

If a message is received, this parameter contains the name of the job that sent the message.

# SENDUSER:

If a message is received, this parameter contains the user name of the job that sent the message.

## **SENDJOBNBR**:

If a message is received, this parameter contains the job number of the job that sent the message.

## **SENDDATE**:

If a message is received, this parameter contains the date that the message was sent. The date is in ISO format - that is, CCYY-MM-DD.

## SENDTIME:

If a message is received, this parameter contains the time that the message was sent. The time is in ISO format - that is, HH:MM:SS.

# MSGQ\_SEND

This activity sends a message to a named message queue on the IBM i server. It is supported only on IBM i servers.

You can use this activity to send an impromptu message or a pre-defined message.

An impromptu message is one for which the message text is entirely supplied at run-time. To send an impromptu message, specify the message text in the MSG parameter value. The MSGID, MSGF, MSGFLIB and MSGDTA parameter values are ignored if MSG is specified.

A pre-defined message uses a message description contained in a message file to provide the text for the message. To send a pre-defined message, do not provide a value for the MSG parameter but instead specify the message identifier in the MSGID parameter and specify the message file that contains the message description in the MSGF parameter (and the MSGFLIB parameter, if necessary). When using a pre-defined message you may also provide message substitution values using the MSGDTA parameter for messages that are defined to use them.

The message is always sent with a message type of \*INFO.

Refer also to the companion MSGQ\_RECEIVE and WATCH\_MSGQ activities that allow you to receive messages from a named message queue. Refer to IBM i documentation concerning message queues, message descriptions and message files for further information about the operating system features upon which this activity depends.

# **INPUT Parameters:**

## TOMSGQ : Required

This parameter must specify the name of the message queue to which the message is sent.

# TOMSGQLIB: Optional

This parameter may specify the name of the library containing the message queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named message queue.

# MSG: Optional

This parameter may specify the message text for an impromptu message. If message text is specified in this parameter, any values specified in the MSGID, MSGF, MSGFLIB and MSGDTA parameters are ignored.
### **MSGID**: Optional

Specifies the message identifier for a pre-defined message description contained in the message file named in the MSGF parameter. The message identifier is an alphanumeric value of seven characters and numbers. For example, CPF9898 is the message identifier for a general-purpose message description supplied with the IBM i operating system in message file QCPFMSG. This value is ignored if impromptu message text is specified in the MSG parameter.

### MSGF: Optional

Specifies the name of the message file containing the pre-defined message description identified by the MSGID parameter. For example, QCPFMSG is the name of a message file supplied with the IBM i operating system, but you may use message files of your own as well. This value is ignored if impromptu message text is specified in the MSG parameter.

### MSGFLIB: Optional

This parameter may specify the name of the library containing the message file identified by the MSGF parameter. If not specified, a default of \*LIBL is used which means the library list is used to locate the named message file. This value is ignored if impromptu message text is specified in the MSG parameter.

### **MSGDTA:** Optional

When using a pre-defined message, this parameter may specify a string containing one or more substitution values that are used as message data fields within the pre-defined message. This value is ignored if impromptu message text is specified in the MSG parameter.

#### **OUTPUT Parameters:**

There are no output parameters

## NOTIFYEVENT

This activity can be used to trigger any event that is defined to LANSA Composer (in event maintenance).

Usually events are triggered automatically by the activities that are associated with them. However, using this activity, you can define your own events (or re-use the supplied event definitions) and trigger them explicitly where appropriate in your processing sequences.

#### **INPUT Parameters:**

#### **EVENTID** : Required

This parameter must contain the name/identifier of an event defined in Event Maintenance. This can be one of ths shipped events or any event that you defined.

#### **OUTPUT Parameters:**

None.

### NEXTNUMBER

This activity generates the next number in a series that is identified by the two "key" values that can be specified in the input parameters. It may be suitable in some applications for generating sequential batch numbers or similar for processed data.

The series from which the number is generated is determined by the two number "keys" specified in the parameters. You can use the keys to generate a sequence number identified appropriately for your purpose or application. For example, you might specify "ORDER\_IN" as the first key and "BATCHNUMBER" as the second key. The first time the activity runs (in any processing sequence) with these keys, the number 1 (one) is returned. Each subsequent invocation with these keys (in any processing sequence) returns the next number in the ORDER\_IN/BATCHNUMBER series.

The numbers generated have a maximum precision of 30, 0. When the number reaches its maximum capacity it will roll-over to zero. The numbers and the keys are held in a Composer database file.

This activity may not meet the standards for data integrity and security required for some applications. You should consider the requirements of your application carefully before using this activity.

### **INPUT Parameters:**

### NUMBERKEY1 : Optional

This is the first of two optional "keys" that are used to uniquely identify the series from which the next sequential number will be generated. If this parameter is not specified, the activity will use the internal identifier of the containing processing sequence (a unique 32 character string). This results in a series of numbers unique to that processing sequence. But two instances of the activity in the same processing sequence would generate numbers from the same series (unless you specify a value for the NUMBERKEY2 parameter).

### NUMBERKEY2 : Optional

This is the second of two optional "keys" that are used to uniquely identify the series from which the next sequential number will be generated. This parameter has no default - if not specified, a blank key value will be used.

### **OUTPUT Parameters:**

NUMBEROUT :

Upon completion, this parameter contains the next sequential number in the series identified by the specified (or assumed) "keys".

### NULL

This activity does nothing.

Clearly this isn't useful in a production environment, but it can sometimes be useful as a placeholder during the design of your processing sequence.

### **INPUT Parameters:**

None.

### **OUTPUT Parameters:**

None.

## PATHMAKE

This activity receives a folder path and a file name and extension and constructs and returns the full file path.

The activity's behavior is subject to the operating environment. For example, it uses or looks for a path delimiter of '\' or '/' according to whether the processing sequence engine is running on a Windows or IBM i server.

### **INPUT Parameters:**

### **PATHIN** : Required

This parameter specifies the folder path - that is the portion of the path left of the file name and extension. It may include a drive name or network location. It may or may not include the terminating folder separator ('\' or '/' according to operating environment). If it does not, the activity will append it.

#### FILEIN : Optional

This parameter specifies the file name. It may include the file extension, in which case the EXTENSIONIN parameter should not be specified.

#### **EXTENSIONIN** : Optional

This parameter specifies the file extension, if required. If specified, the activity will append this to the file path. It may or may not include the period (.) separator - if it does not, the activity will add it.

# **OUTPUT Parameters:**

### **PATHOUT** :

Upon completion, this parameter will contain the full file path.

## PATHSPLIT

This activity receives a file path and extracts the constituent components including the folder path, file name and extension.

The activity's behavior is subject to the operating environment. For example, it uses or looks for a path delimiter of '\' or '/' according to whether the processing sequence engine is running on aWindows or IBM i server.

### **INPUT Parameters:**

### **PATHIN** : Required

This parameter specifies the file path to be split.

### **OUTPUT Parameters:**

### **PATHOUT**:

Upon completion, this parameter will contain the folder path - that is the input file path minus the file name and extension.

### FILEOUT :

Upon completion, this parameter will contain the file name if one is found in the input file path.

### FILEXOUT :

Upon completion, this parameter will contain the file name and extension if one is found in the input file path.

### **EXTENSIONOUT**:

Upon completion, this parameter will contain the file extension (not including the period) if one is found in the input file path.

## PUT\_DTAARA

This activity writes a value to the specified data area. It is supported only on IBM i servers.

This activity can only write to data areas created with TYPE(\*CHAR) and the starting position and length specified must be valid for the definition of the data area specified. The data area must exist and the job running the activity must have the necessary authorities.

#### **INPUT Parameters:**

#### **DTAARA** : Required

This parameter specifies the name of the data area.

LIB : Optional

This parameter can specify the name of the library containing the data area. If it is not specified, the processing sequence will use the library list to locate the data area.

#### **START** : Required

This parameter specifies the starting position in the data area to which to write the value.

**LENGTH** : Required

This parameter specifies the length of the value to be written.

### **STRINGIN** : Required

This parameter specifies the value to be written to the specified positions in the specified data area.

### **OUTPUT Parameters:**

There are no output parameters

## QUERY\_CCSID

This activity returns the IBM i CCSID for the specified file. The activity is only supported on IBM i servers.

A CCSID is a coded character set identifier that identifies a specific encoding of a specific code page on IBM i servers and other IBM platforms and software. It is crucial that a file containing textual data has the correct CCSID that represents its encoding of that data – otherwise programs cannot successfully interpret the file. For more information on CCSIDs, refer to relevant IBM documentation.

### **INPUT Parameters:**

### FILEPATH : Required

This parameter should contain the full name and path of the file whose CCSID is to be queried.

### **OUTPUT Parameters:**

CCSID :

If successful, this parameter will contain the CCSID for the specified file.

## RANDOMNUMBER

This activity generates a pseudo-random number in the range specified by the LOWERBOUND and UPPERBOUND input parameters. For example, if the default values of 1 (one) and 100 (one hundred) are used for the LOWERBOUND and UPPERBOUND parameters, the resulting pseudo-random number will be in the range 1 to 100 inclusive.

Typically (but not invariably), each use of the RANDOMNUMBER activity will yield a result different to the previous and frequent use will yield results that will be evenly spread, on average, over the range specified by the LOWERBOUND and UPPERBOUND input parameters.

**Technical note:** The implementation of the RANDOMNUMBER activity uses the LANSA RANDOM\_NUM\_GENERATOR built-in function with a stream index value of 77. Refer to LANSA documentation for more information on this built-in function.

### **INPUT Parameters:**

### LOWERBOUND : Optional

This optional parameter specifies the lower bound of the range of the resulting pseudo-random number. If not specified, a default value of 1 (one) is used.

### **UPPERBOUND** : Optional

This optional parameter specifies the upper bound of the range of the resulting pseudo-random number. If not specified, a default value of 100 (one hundred) is used.

### **OUTPUT Parameters:**

NUMBEROUT :

Upon completion, this parameter contains the generated pseudo-random number in the range specified by the LOWERBOUND and UPPERBOUND input parameters.

## **RENAME\_FILE**

This activity will rename a file.

When executed on a Windows platform a system rename command is executed. When executed on an IBM i platform an i5/OS REN command is executed.

### **INPUT Parameters:**

### FROMFULLNAME : Required

This parameter should contain the full path and name of the file to be renamied.

For example: Windows C:\dir1\myfile.txt

IBM i /indir/myfile.txt

**TONAME** : Required

This parameter should contain the new name (and optional extension) of the file. Note that you cannot specify a new drive or path for the destination file. The file is renamed in the same directory containing the existing object.

### **OUTPUT Parameters:**

There are no output parameters.

## SAVE\_PSVSET

This activity saves processing sequence variables and their values to a PSV file. The variables and their values may subsequently be loaded into the same or a different processing sequence using the LOAD\_PSVSET activity or transformed using a Transformation Map. Built-in variables (and other variables used internally by LANSA Composer) are not saved.

If no processing sequence variables are selected, the activity completes normally and a valid PSV file is created (containing no <psVariable> elements). The value of the PSVCOUNT output parameter will be zero in this case.

The PSV file created by this activity will always contain a single PSV set (<psvSet> element). However, PSV files created by other means (for example, as the output from a Transformation Map) may contain more than one PSV set (processing sequence variables set) and the LOAD\_PSVSET activity is capable of processing files containing one or more PSV sets.

For more information about PSV files, refer to the following.

Saving, Loading and Transforming Processing Sequence Variables

#### **INPUT Parameters:**

#### **PSVFILEPATH** : Required

This parameter must specify the path and file name of the PSV file to be created. By convention, we suggest using a file extension of '.PSV', though this is not a requirement. If the specified file already exists, it will be replaced by this activity.

### **PSVSELECT** : Optional

This parameter may optionally specify a generic name of the processing sequence variables to be saved. For example, specify 'ORDER' to save the processing sequence variables whose names begin with the characters 'ORDER'. Note that processing sequence variable names are case-insensitive. It does not matter whether you specify the generic name using upper or lowercase characters, or a mixture of both. If the generic name specified ends with '\*' (a common "generic" notation), the trailing asterisk is removed before performing the comparisons. If this parameter is not specified, a default value of '\*ALL' is assumed. Values of '\*', '\*ALL' or blank will result in all processing sequence variables being selected.

### **PSVSETID** : Optional

This parameter may optionally specify a value to be written to the 'id='

attribute of the <psvSet> element in the PSV file. The value of this atribute will be available, if required, to the processing sequence that loads the variable set using the LOAD\_EACHPSVSET activity.

### **PSVCOMMENT** : Optional

This parameter may optionally specify a value to be written to the 'comment=' attribute of the <psvSet> element in the PSV file. The value of this atribute will be available, if required, to the processing sequence that loads the variable set using the LOAD\_EACHPSVSET activity.

#### OUTPUT Parameters: PSVCOUNT:

Upon successful completion, this output parameter will contain a count of the processing sequence variables saved.

## SLEEP

This activity will suspend processing for a time interval specified by the INTERVAL parameter. When the specified time interval has elapsed, the activity ends and processing continues with the next directive in the processing sequence.

When used on LANSA Composer IBM i Server, the specified interval is rounded up to the nearest 1000 milliseconds (one second).

#### **INPUT Parameters:**

#### **INTERVAL** : Optional

This parameter specifies the interval in milliseconds to suspend processing. If not specified, the activity assumes a default of 1000 milliseconds (one second). When used on LANSA Composer IBM i Server, the value specified is rounded up to the nearest 1000 milliseconds (one second).

### **OUTPUT Parameters:**

There are no output parameters

## SMS\_SEND

This activity will send an SMS to a mobile number using an e-mail-based service provided for the purpose by a third party SMS provider. Such services require you to hold an account with them (usually paid). Once you have an account, your application sends an e-mail to the provider in a specified format. The SMS\_SEND activity uses an e-mail format used by www.streetdata.com.au and other providers in which the key features are:

- the telephone number for the receiver is specified left of the '@' sign in the e-mail address (for example, 04001234567@streetdata.com.au)
- the subject specifies the account holder's assigned user name and password
- the e-mail body contains the message text to be sent

To use this Activity you must have an SMTP mail server available and an SMS provider. If your SMS provider expects email in a format different to that generated by this Activity, then the activity processor for the activity may need to be modified to accommodate it.

Details for connecting to and using the SMS provider service and for the SMS message destination and content may be specified in an 2.3.7 SMS Configuration. Refer to the description of 2.3.7 SMS Configuration for additional information.

### **INPUT Parameters:**

### **SMSCONFIGID** : Optional

This parameter may contain the name of an SMS Configuration to be used to send the SMS message. This Configuration can be created and maintained using the SMS Configuration. If the specified SMS configuration is not found or not active, the Activity will return an error and processing will be abandoned. If the SMS configuration name is not provided then the default SMS configuration set in System Settings will be used.

### **SMSTONUM** : Optional

This parameter may contain the SMS number to which the message is to be sent. If specified here, it overrides the value specified in the SMS configuration. If not specified here, then the SMS number must be specified in the SMS configuration.

**SMSMSG** : Optional

This parameter may contain the SMS message text to be sent. If specified here, it overrides the value specified in the SMS configuration. If not specified here, then the SMS message text must be specified in the SMS configuration. Most SMS providers restrict the SMS message length to a maximum number of characters, for example 160 characters. It is your responsibility to ensure the message text does not exceed the maximum imposed by the service you are using.

#### **OUTPUT Parameters:**

There are no output parameters.

## SORT\_LISTS

This activity will sort one or more variable lists in "parallel" into the order of the values in the list specified by the SORTLIST1 parameter.

In the simplest case, where you want to sort one list, simply specify the required variable list on the SORTLIST1 parameter.

You might specify more than one list in cases where you have variable lists that are linked. For example, you have two variable lists ORDERS and CUSTOMERS that contain an order number and customer number respectively for a list of ORDERS. For every entry in ORDERS, there is also an entry in CUSTOMERS that contains the customer number for the corresponding order.

To sort the two lists in "parallel" into the sequence of the order numbers, you could specify ORDERS as the list for the SORTLIST1 parameter and CUSTOMERS as the list for the SORTLIST2 parameter.

If you specify more than one list, all the lists must have the same number of list items. Otherwise, the activity will end in error.

NOTE: This activity imposes a maximum number of list items that can be sorted. If the variable list contains more than the maximum, the activity will end in error. The maximum is usually 9,999. However, using variable lists of this extent is generally NOT recommended in the context of this activity or any other aspect of LANSA Composer variable list processing.

### **INPUT Parameters:**

### **SEQUENCE** : Optional

This parameter specifies whether the variable list(s) should be sorted in ascending (\*ASCEND) or descending (\*DESCEND) order of the values in the list specified by the SORTLIST1 parameter.

### **INPUT and Output Parameters:**

### **SORTLIST1** : Required

Specifies the first variable list to be sorted. If more than one list is specified, it is the values in THIS list that are used to sort ALL the lists in parallel. This parameter must be specified.

#### SORTLIST2SORTLIST3 SORTLIST4 SORTLIST5 : Optional

You may specify up to four additional variable lists to be sorted in "parallel". If you use these parameters you must specify them contiguously -

LANSA Composer will stop looking after the first unused parameter. Each list specified here must have exactly the same number of list items as the list specified in the SORTLIST1 parameter - otherwise the activity will end in error.

## SPLF\_LIST

This activity fills the output lists with details of all or selected spool files in a specified output queue. Spool files in the output queue can be selected by user, user data or form type by specifying values for the corresponding parameters.

Entries in each output list correspond by index to entries in the other lists for the same spool file.

Activities that populate list variables are often followed by a LOOP processing sequence directive or other constructs to process the contents of the list. Refer to Variables and Lists for more information about the use of list variables.

### **INPUT Parameters:**

### **OUTQ** : Required

This parameter specifies the name of the output queue from which the spool files are listed.

### **OUTQLIB** : Optional

This parameter specifies the library containing the output queue from which the spool files are listed. If not specified, a default value of \*LIBL is used, meaning the library list is searched to find the output queue named in the OUTQ parameter.

### **SELECTUSER** : Optional

If this parameter is specified, the output lists will include only the spool files for the specified user (that also meet other selection criteria). If not specified, spool files for all users are eligible to be included.

### SELECTUSERDATA : Optional

If this parameter is specified, the output lists will include only the spool files whose user data (USRDTA) matches the user data specified (and that also meet other selection criteria). If not specified, all spool files are eligible to be included irrespective of their user data.

### **SELECTFORMTYPE** : Optional

If this parameter is specified, the output lists will include only the spool files whose form type (FORMTYPE) matches the form type specified (and that also meet other selection criteria). If not specified, spool files of any form type are eligible to be included.

#### OUTPUT Parameters: SPLFNAMELIST :

Upon successful completion, this parameter will contain a list of the spool file names for spool files that were selected from the specified output queue.

### **SPLFNUMBERLIST** :

Upon successful completion, this parameter will contain a list of the spool file numbers for spool files that were selected from the specified output queue.

### **JOBNAMELIST** :

Upon successful completion, this parameter will contain a list of the job names for spool files that were selected from the specified output queue. The job names apply to the job that created the spool files.

### **JOBUSERLIST** :

Upon successful completion, this parameter will contain a list of the user profile names for spool files that were selected from the specified output queue. The user profile names apply to the job that created the spool files.

### JOBNUMBERLIST :

Upon successful completion, this parameter will contain a list of the job numbers for spool files that were selected from the specified output queue. The job numbers apply to the job that created the spool files.

## SPLF\_TO\_PDF

This activity converts the spooled file identified by the job and spool file input parameters into a PDF document. The PDF document is written to the file name and path specified by the DOCPATH parameter.

Additional input parameters allow basic formatting and layout of the PDF document to be specified. The MOREATTRIBUTES parameter is provided for advanced use to specify additional PDF document attributes.

#### **INPUT Parameters:**

### **DOCPATH** : Required

This parameter specifies the path and file name for the PDF document that will be generated by this activity. If the specified file already exists, by default it's contents will be replaced, but this behavior can be altered by the APPEND parameter.

### **APPEND** : Optional

This optional parameter only applies if the specified PDF document path and file name already exists. It takes a value of Y (yes) or N (no). If Y (yes) is specified, the converted spool file pages will be appended to the existing PDF document. If N (no) is specified or assumed, the existing PDF document will be replaced. If this parameter is not specified, a default of N (no) is assumed.

### **JOBNAME** : Required

This parameter specifies the name of the job that created the spool file from which the PDF document is to be created.

### JOBUSER : Required

This parameter specifies the user profile name of the job that created the spool file from which the PDF document is to be created.

### JOBNUMBER : Required

This parameter specifies the job number of the job that created the spool file from which the PDF document is to be created.

### **SPLFNAME** : Required

This parameter specifies the name of the spool file from which the PDF document is to be created.

### **SPLFNUMBER** : Optional

This parameter specifies the number of the spool file from which the PDF

document is to be created. If not specified, a default value of 1 is used. Otherwise specify the number of the job's spooled file from which the PDF document is to be created.

### **PAGES** : Required

This parameter can specify a comma-separated list of page numbers and/or page ranges that are to be included in the conversion to PDF. For example the value '2,5,10-15' will select single pages 2 and 5 and pages 10 to 15. If this parameter is not specified, all pages in the spool file will be converted to PDF.

### **PAGETYPE** : Required

According to the value of this parameter, the converted PDF document may consists of PDF text pages or PDF TIFF G4 image pages. The value \*TEXT may be used for SCS type spooled files and \*IMAGE may be used for both SCS and AFPDS type spooled files. \*TEXT is faster but can only convert SCS spool files. \*IMAGE is slower but can be used for both SCS and AFPDS file types. If this parameter is not specified, \*TEXT is assumed.

### **PAGESIZE** : Optional

Specifies the page size for the generated PDF document. The default value is \*A4. Other valid values include:

- \*A0 to \*A10
- \*B0 to \*B5
- \*LETTER
- \*HALFLETTER
- \*LEDGER
- \*LEGAL
- \*NOTE

### **ORIENTATION** : Optional

Specifies the page orientation as \*PORTRAIT or \*LANDSCAPE. The default value is \*LANDSCAPE.

### BACKIMAGEPATH : Optional

This parameter may specify the path and file name of an image file (such as a corporate logo or letterhead) to be included in the background of the generated PDF file. If specified, the image is included and positioned according to the values of the BACKIMAGEPAGES and BACKIMAGEPOS parameters. When this feature is used, the image is placed on the PDF page and the spool file text is placed over it. The position or layout of the spool file text is not altered to accommodate the image. Therefore, you should choose an image file and/or position or size the image to avoid obscuring the spooled file text. For example, choose an image file that uses mainly light colours.

### BACKIMAGEPAGES : Optional

This parameter applies only when a background image file is specified using the BACKIMAGEPATH parameter. It specifies the pages of the PDF file that are to include the image. You can use the value \*ALL to include the image on every page of the generated PDF file, or you can use the value \*FIRST to include the image only on the first page of the PDF file. The default value is \*ALL.

### BACKIMAGEPOS : Optional

This parameter applies only when a background image file is specified using the BACKIMAGEPATH parameter. It specifies the size and position of the image on each affected page of the PDF file. It must be specified as four numeric values separated by commas. The four values specify respectively the x (left) and y (top) coordinates and the width and height of the image. All values are specified in *points* – each *point* is 1/72 inches. If not specified, the default values 1, 1, 80, 80 are used.

### **MOREATTRIBUTES** : Optional

This parameter is provided for advanced use. It allows additional PDF document attributes to be specified using the syntax:

### keyword(value) {keyword(value) ...}

Refer to LANSA Integrator documentation for the CREATE service command of the PDFSpoolFileService for details of the keywords and values that may be specified.

### **OUTPUT Parameters:**

There are no output parameters.

## SPLF\_TO\_TEXT

This activity will convert the spooled file identified by the job and spool file input parameters into a text document. The text document is written to the file name and path specified by the DOCPATH parameter. The FORMAT parameter allows you to choose the format that is applied to the spooled file data when it is written to the text document.

Amongst other things, you may be able to use this activity to mine data contained in spooled files by combining the activity with a Transformation Map that parses and extracts information from the generated text document using the FlexText component of the mapping tool.

The implementation of this activity uses the IBM i CPYSPLF command. Some spooled files, for example spooled files that contain only advanced function printing data stream (AFPDS) data cannot be processed by the CPYSPLF command. Refer to IBM i documentation of the CPYSPLF command for more information.

### **INPUT Parameters:**

### **DOCPATH** : Required

This parameter specifies the path and file name for the text document that will be generated by this activity. If the specified file already exists, its contents will be replaced.

### **JOBNAME** : Required

This parameter specifies the name of the job that owns the spool file from which the text document is to be created.

### JOBUSER : Required

This parameter specifies the user profile name of the job that owns the spool file from which the text document is to be created.

### JOBNUMBER : Required

This parameter specifies the job number of the job that owns the spool file from which the text document is to be created.

### **SPLFNAME** : Required

This parameter specifies the name of the spool file from which the text document is to be created.

### SPLFNUMBER : Optional

This parameter specifies the number of the spool file from which the text

document is to be created. It is required only if there is more than one spool file of the name specified by the SPLFNAME parameter for the job specified. If not specified, a default special value of \*ONLY is used. You may specify special values \*ONLY or \*LAST. Otherwise specify the number of the job's spooled file that is to be converted.

### **DOCCCSID** : Optional

This parameter allows you to specify the IBM i CCSID with which the output text document is created.

If the parameter is not specified, a default of \*DEFAULT is assumed, which instructs the activity to create the output text document using the CCSID for the job in which the activity is executing. (You can also specify the special value of \*JOB which has the same effect.)

Otherwise, you should specify the numeric CCSID value required. For example, a value of 1208 means UTF-8. Refer to IBM i documentation for a complete list and description of the available CCSIDs.

NOTE: the assumed or explicit CCSID is applied only when the specified output file does NOT already exist. If the specified output file already exists and is being replaced or appended-to by this activity, then its CCSID will not be changed.

### FORMAT : Optional

This parameter allows you to choose the format that is applied to the spooled file data when it is written to the text document. You may specify one value from each of the following three groups (each value should be separated by at least one space):

- 1. \*SPOOLDATA | \*PRTCTL | \*PAGELINE
- 2. \*FIXED | \*CSV

### 3. \*BLANKLINES | \*NOBLANKLINES | \*CONDENSEBLANKLINES

You must specify quote marks around the value(s) to distinguish them from built-in variable names.

The options in each of the three groups are described further below. If you specify more than one value from each group, the last-specified value is effective. If you do not specify this parameter, the default values are '\*SPOOLDATA \*FIXED \*BLANKLINES'.

### 1. \*SPOOLDATA | \*PRTCTL | \*PAGELINE

These values specify what data items to include in each output line. The default is \*SPOOLDATA, which includes only the spooled file line contents.

Specify \*PRTCTL to include skip before and space before values before each spooled file line contents. Specify \*PAGELINE to include page number and line number values before each spooled file line contents. Note that the page and line numbers are calculated. In some cases, they may not match exactly the page and line numbers in the original spooled file.

### 2. \*FIXED | \*CSV

These values specify how the data items on each output line are formatted. The default is \*FIXED, which includes the data items in fixed-length columns.

For FORMAT('\*PRTCTL \*FIXED'), the skip before value will occupy columns 1-3, the space before value will occupy column 5 and the spooled file line contents will begin in column 7.

For FORMAT('\*PAGELINE \*FIXED'), the page number value will occupy columns 1-7, the line number value will occupy columns 9-11 and the spooled file line contents will begin in column 13.

Specify \*CSV to have the data items output in comma-separated form instead.

For FORMAT('\*SPOOLDATA'), the \*FIXED or \*CSV formatting values do not apply and have no effect.

3. \*BLANKLINES | \*NOBLANKLINES | \*CONDENSEBLANKLINES

These values specify whether blanks lines are output to the text document. The default is \*BLANKLINES which will include all blank lines that occur in the spooled file up to the last non-blank line on each page.

Specify \*NOBLANKLINES to have blank lines suppressed altogether. Note that this option does not change the generated page and line number values written for the non-blank lines (that is to say, the blank lines are still counted for this purpose).

Specify \*CONDENSEBLANKLINES to have only the first in each group of consecutive blank lines written to the text document. Again, this option does not change the generated page and line number values written for the non-blank lines (that is to say, the blank lines are still counted for this purpose).

### **OUTPUT Parameters:**

There are no output parameters.

## SQL\_CALL

This activity executes a stored procedure on the target database. Use this activity when your stored procedure does not return a result set. It may, optionally return a single result that can be received in the SQLRESULT output parameter of this activity.

If your stored procedure returns a result set, then you should use the SQL\_CALLQUERY or SQL\_CALLQRYCSV activity instead.

This activity supports the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If your solution uses this capability, you must first execute one of the following activities to supply the parameter value(s):

SQL\_PARAMS Set parameter values for SQL operation

SQL\_PARAMSCSV Set parameter values for SQL operation from CSV

Note that only input (IN) parameters for an SQL stored procedure are presently supported. You cannot pass or receive OUT and INOUT stored procedure call parameters.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

### **SQLCALL** : Required

This parameter must specify the name of the stored procedure (which may be qualified with a library or schema name) along with any parameters (or parameter markers) that it requires. The *CALL* keyword should not be included. It should usually be in the form:

library or schema>.procedure(<parameters>)

### OR

library or schema>/procedure(<parameters>)

(depending on the syntax used by your database and JDBC driver.)

The SQL statement may make use of parameter markers (usually designated by a question mark). You must use either the SQL\_PARAMS or the SQL\_PARAMSCSV activity *before* this activity, to supply the parameter values. Note that the SQL\_CALL activity will make use of only one set (or "row") of parameter values.

The following is an example that might be specified in this parameter to call the SQL stored procedure DXTUTSQL01, passing it two parameter values that have previously been supplied using either the SQL\_PARAMS or the SQL\_PARAMSCSV activity:

DXTUTSQL01(?, ?)

### SQLRETURNING : Optional

This parameter specifies the expected data type of the return value, if any, from the stored procedure. You can choose from the following values:

\*NONE \*CHAR \*STRING \*SMALLINT \*INTEGER \*FLOAT \*DOUBLE \*DECIMAL \*NUMERIC

However, the datatypes possible depend on what your target database and JDBC driver can support. And, in any particular instance, of course, it depends on the definition of the stored procedure.

The default value is \*INTEGER. If your database is on an IBM i server, note that presently the IBM i database only supports a data type of integer for values returned from a stored procedure. If your stored procedure does not return a value, then you can specify \*NONE to signify this.

### **OUTPUT Parameters:**

## SQLRESULT :

If successful, this output parameter will contain the return value, if any, from the stored procedure.

## SQL\_CALLQRYCSV

This activity executes a stored procedure on the target database, receives a result set and writes the result set directly into the specified file in CSV format.

If your stored procedure does not return a result set, then you should use the SQL\_CALL activity instead. Refer also to the description of the SQL\_CALLQUERY activity for an alternate way to receive the result set from a stored procedure call.

This activity supports the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If your solution uses this capability, you must first execute one of the following activities to supply the parameter value(s):

SQL\_PARAMS Set parameter values for SQL operation

SQL\_PARAMSCSV Set parameter values for SQL operation from CSV

Note that only input (IN) parameters for an SQL stored procedure are presently supported. You cannot pass or receive OUT and INOUT stored procedure call parameters.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

### **SQLCALL** : Required

This parameter must specify the name of the stored procedure (which may be qualified with a library or schema name) along with any parameters (or parameter markers) that it requires. The *CALL* keyword should not be included. It should usually be in the form:

library or schema>.procedure(<parameters>)

### OR

library or schema>/procedure(<parameters>)

(depending on the syntax used by your database and JDBC driver.)

The SQL statement may make use of parameter markers (usually designated by a question mark). You must use either the SQL\_PARAMS or the SQL\_PARAMSCSV activity *before* this activity, to supply the parameter values. Note that the SQL\_CALLQRYCSV activity will make use of only one set (or "row") of parameter values.

The following is an example that might be specified in this parameter to call the SQL stored procedure DXTUTSQL01, passing it two parameter values that have previously been supplied using either the SQL\_PARAMS or the SQL\_PARAMSCSV activity:

DXTUTSQL01(?, ?)

### **SQLFILEPATH** : Required

This parameter must specify the path and file name of the CSV file to be created or appended by this activity. If the file already exists (and NO is specified or assumed for the SQLAPPEND parameter), then the file will be overwritten by this activity.

### **SQLAPPEND** : Optional

If the specified output file already exists, then you can specify YES for this parameter to cause the activity to append the new results to the end of the existing contents of the file. Otherwise the file is replaced with the new contents. If the specified output file does not already exist, then this parameter is ignored.

### SQLCCSID : Optional

This parameter applies only on IBM i servers. It allows you to specify the IBM i CCSID with which the output CSV file is created.

If the parameter is not specified, a default of \*DEFAULT is assumed, which instructs the activity to create the output text file using the CCSID for the job in which the activity is executing. (You can also specify the special value of \*JOB which has the same effect.)

Otherwise, you should specify the numeric CCSID value required. For example, a value of 1208 means UTF-8. Refer to IBM i documentation for a

complete list and description of the available CCSIDs.

NOTE: the assumed or explicit CCSID is applied only when the specified output file does NOT already exist. If the specified output file already exists and is being replaced or appended-to by this activity, then its CCSID will not be changed.

### SQLCOLUMNS : Optional

This parameter must specify the number of columns from the expected result set that are to be written to the CSV file, up to a maximum of 25. This, of course, will depend on the stored procedure that you are calling.

You may specify fewer columns than returned from your stored procedure call. However, if you specify more, then a run-time error will occur, for example

### ERROR – Descriptor index not valid, or ERROR – The index 10 is out of range

### **COLSEPARATOR** : Optional

You may specify the separator character that is used to separate the column values in each line of the CSV output file. The most common form of CSV uses a comma as the separator, and that is the default value for this parameter if you do not specify it.

If a non-comma separator is specified, it should be 1 character in length and can consist of any character.

### **COLHEADINGS** : Optional

If you wish the first row of the output CSV file to contain comma-separated column headings, you may specify the heading line in this parameter. Note you should specify the entire string, including the separators, as a single value – for example:

Heading1, Heading2, Heading3

If you do not specify a value for this parameter, then no heading row will be written to the CSV file. If you specified YES for the SQLAPPEND parameter, then the COLHEADINGS value is not used.

### SQLMAXROWS : Optional

This parameter may specify a maximum number of rows to be returned. This guards against the possibility of stored procedure calls that select much more data than was intended.

If not specified, a default value of 100 (one hundred) is used. Remember

that the SQL database activities are not intended and not usually suitable for high-throughput, high-volume database operations. If you expect that your stored procedure will return a large number of records, then you should possibly consider an alternate implementation, such as using a Transformation Map.

#### OUTPUT Parameters: SQLROWS :

Upon successful completion, this parameter will contain the actual number of rows returned by the stored procedure call and written to the CSV file.

## SQL\_CALLQUERY

This activity executes a stored procedure on the target database. Use this activity when your stored procedure returns a result set.

If your stored procedure does not return a result set, then you should use the SQL\_CALL activity instead. Refer also to the description of the SQL\_CALLQRYCSV activity for an alternate way to receive the result set from a stored procedure call.

This activity supports the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If your solution uses this capability, you must first execute one of the following activities to supply the parameter value(s):

SQL\_PARAMS Set parameter values for SQL operation

SQL\_PARAMSCSV Set parameter values for SQL operation from CSV

Note that only input (IN) parameters for an SQL stored procedure are presently supported. You cannot pass or receive OUT and INOUT stored procedure call parameters.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

### **SQLCALL** : Required

This parameter must specify the name of the stored procedure (which may be qualified with a library or schema name) along with any parameters (or parameter markers) that it requires. The *CALL* keyword should not be included. It should usually be in the form:

library or schema>.procedure(<parameters>)

### OR

library or schema>/procedure(<parameters>)

(depending on the syntax used by your database and JDBC driver.)

The SQL statement may make use of parameter markers (usually designated by a question mark). You must use either the SQL\_PARAMS or the SQL\_PARAMSCSV activity *before* this activity, to supply the parameter values. Note that the SQL\_CALLQUERY activity will make use of only one set (or "row") of parameter values.

The following is an example that might be specified in this parameter to call the SQL stored procedure DXTUTSQL01, passing it two parameter values that have previously been supplied using either the SQL\_PARAMS or the SQL\_PARAMSCSV activity:

DXTUTSQL01(?, ?)

### SQLMAXROWS : Optional

This parameter may specify a maximum number of rows to be returned. This guards against the possibility of stored procedure calls that select much more data than was intended.

If not specified, a default value of 100 (one hundred) is used. Remember that the SQL database activities are not intended and not usually suitable for high-throughput, high-volume database operations. If you expect that your stored procedure will return a large number of records, then you should possibly consider an alternate implementation, such as using a Transformation Map.

### **OUTPUT Parameters:**

### SQLROWS :

Upon successful completion, this parameter will contain the actual number of rows returned by the stored procedure call.

### SQLCOLUMN1 SQLCOLUMN2

•••

### SQLCOLUMN25

These output lists will contain a list of the values for the corresponding column for each row returned by the stored procedure call.

You should specify the name of a variable list that will contain the values for each column returned by your stored procedure, up to a maximum of 25. You may specify fewer output lists than returned from your stored procedure call. However, if you specify more, then a run-time error will occur, for example

*ERROR – Descriptor index not valid, or ERROR – The index 10 is out of range*
# SQL\_COMMIT

This activity commits a database transaction performed by one or more prior SQL database activities such as SQL\_UPDATE.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

#### **INPUT Parameters:**

#### SQLHANDLE : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

#### **OUTPUT Parameters:**

# SQL\_CONNECT

This activity establishes a connection to an SQL database and returns a "handle" that can be used to identify this database connection for use in other SQL database activities.

Typically in LANSA Composer, most database activity is accomplished through the use of Transformation Maps. However, this activity along with its related SQL database activities provides another option for performing limited database query and update operations on an SQL database, as well as a means of invoking SQL stored procedures in the database.

The SQL\_CONNECT activity establishes the database connection using a database configuration whose name you provide in the DBCONFIG parameter. The database configuration contains the implementation-specific database connection information and user credentials. Because the connections use JDBC, like transformation maps, the SQL activities are capable of addressing any compatible database that is network-addressable from the server system running LANSA Composer, if a suitable JDBC driver is available. This means that a LANSA Composer system running on an IBM i server, for example, could address an SQL server database running on a Windows server in the same network, or vice-versa.

More than one SQL database connection may be active at one time in a single Processing Sequence.

The SQL database activities are not intended and not usually suitable for highthroughput, high-volume database operations. Rather they provide a simple means to complete a business process integration solution that may not otherwise have been possible, with some simple, low-volume database access and/or maintenance.

NOTE: LANSA Composer does not guarantee that any form of SQL statement that is valid for your target database can successfully be executed through the SQL database activities, nor that every form of SQL stored procedure can successfully be executed.

Nor will it be possible to successfully address every possible data type in your database. Since processing sequence variables that might be used to pass or receive data to the SQL database are untyped, not all conversions can be successful or yield useful results. It is your responsibility to ensure that any data passed through the SQL database suite of activities is in a form that can be accepted and processed by both the JDBC driver and the target database.

It is recommended that you keep your SQL operations through these activities as simple as possible.

## **Example Processing Sequences using the SQL database activities**

Refer to the following example processing sequences supplied with LANSA Composer for working (\*) examples that use the SQL database activities. (\*) Note that some setup will be required on your system to enable these examples to execute successfully. Refer to the notes accompanying the example processing sequences for details:

- EXAMPLE\_SQL01: Example of using the SQL\_QUERY activity
- EXAMPLE\_SQL02: Example of using the SQL\_UPDATE activity
- EXAMPLE\_SQL03: Example of using the SQL\_CALLQRYCSV activity

## **Related SQL database activities**

The SQL\_CONNECT activity returns a "handle" that can then be passed on to the other SQL database activities to accomplish a range of database tasks. The full suite of SQL database activities are briefly described below:

• Use the following activities to establish or disconnect an SQL database connection:

SQL_CONNECT	Connect to database using SQL
SQL_DISCONNECT	Disconnect from database using SQL

• Use the following activities to query the database:

SQL_QUERY	Query database using SQL
SQL_QUERYTOCSV	Query database using SQL to output CSV file

• Use the following activities to perfom insert, update and delete operations in the database:

SQL\_UPDATEUpdate database using SQL

• Use the following activities to execute an SQL stored procedure in the database:

SQL_CALL	Execute an SQL stored procedure	
SQL_CALLQRYCSV	Query database using an SQL stored procedure to CSV	
SQL_CALLQUERY	Query database using an SQL stored procedure	

• Use the following activities to implement transaction control relating to any database insert, update or delete operations you have performed:

SQL_COMMIT	Commit a database transaction using SQL
SQL_ROLLBACK	Rollback a database transaction using SQL

• Use the following activities to set the parameter values for an SQL operation:

SQL_PARAMS	Set parameter values for SQL operation
SQL_PARAMSCSV	Set parameter values for SQL operation from CSV

## **Eligibility for Processing Sequence Restart**

When a LANSA Composer Processing Sequence run ends in error, it is often

possible to restart it from the point of failure—once the cause of the failure has been corrected. This is a very powerful feature of LANSA Composer.

For a LANSA Composer solution using the SQL database activities, processing sequence restart is supported, but needs to be heavily qualified by exactly what database operations are being performed.

LANSA Composer restart support remembers and can re-establish a previouslyestablished SQL connection, but whether a particular process can effectively be restarted depends on the types of database operations that are being performed and in particular whether they are dependent on earlier SQL database operations that may have completed before the restart.

For example, a process that performs database updates under transaction control may not be restartable in practice, depending on where the failure occurs. On the other hand a solution that exclusively performs SQL query operations will usually be capable of being successfully restarted.

Therefore this decision is left to the solution designer by means of the RESTARTELIGIBLE parameter to the SQL\_CONNECT activity.

If your solution uses the SQL database activities in such a way that restart eligibility cannot be assured for the life of the database connection, then you should specify NO for this parameter.

In any event, to maximize the benefit of LANSA Composer's restart capability, you should complete your SQL database operations and execute the SQL\_DISCONNECT activity at the earliest opportunity. Once the SQL database connection has been closed, normal restart eligibility resumes.

### **INPUT Parameters:**

## **DBCONFIG** : Required

This parameter must specify the name of a 2.3.8 Database Configuration that specifies the details necessary to establish the database connection.

## **RESTARTELIGIBLE** : Optional

This parameter specifies whether the LANSA Composer processing sequence that contains this activity should remain eligible for restart while the SQL database connection remains open. The default value is YES.

If your solution uses the SQL database activities in such a way that restart eligibility cannot be assured for the life of the database connection, then you should specify NO for this parameter.

For more information refer to Eligibility for Processing Sequence Restart above.

### OUTPUT Parameters: SQLHANDLE :

If successful, the value of this output parameter identifies the SQL connection established by this instance of the SQL\_CONNECT activity. The same value must be specified as the SQLHANDLE input parameter value for all subsequent SQL database activities that are to operate on the same SQL database connection.

## SQL\_DISCONNECT

This activity closes an SQL database connection previously established using the SQL\_CONNECT activity.

Once the database connection has been closed, the connection handle value is no longer valid and cannot be used again in further SQL database activities.

If you do not execute the SQL\_DISCONNECT activity for an open SQL database connection before the end of your processing sequence, LANSA Composer will automatically disconnect it. However, it is recommended that you explicitly disconnect using this activity as soon as the SQL database connection is no longer required. For more information on this, refer to:

Eligibility for Processing Sequence Restart

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

## **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## **OUTPUT Parameters:**

## SQL\_PARAMS

This activity sets parameter values to be used subsequently by one of the following SQL database activities:

SQL_CALL	Execute an SQL stored procedure
SQL_CALLQRYCSV	Query database using an SQL stored procedure to CSV
SQL_CALLQUERY	Query database using an SQL stored procedure
SQL_UPDATE	Update database using SQL

The parameter values are used in the SQL statement used by one of the above activities in place of any parameter markers (usually designated by a question mark) that are specified in the SQL statement.

Note that any parameter values that have been set will be cleared after executing any of the above activities or the SQL\_QUERY or SQL\_QUERYTOCSV activities. If you need to use the same set of parameters more than once, then you will need to repeat this activity before each activity that uses the parameters.

Refer also to the description of the SQL\_PARAMSCSV activity, which performs a similar function but permits the parameter values to be supplied from a CSV file.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity.

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## SQLPARAM1SQLPARAM2

... SQLPARAM25: Optional

Use these parameters to supply one or more variable lists, up to a maximum of 25, each "row" of which supplies one set of the parameter values to be used. Each of the lists supplied should have the same number of entries.

A subsequent SQL\_CALL, SQL\_CALLQRYCSV or SQL\_CALLQUERY will use only one set (or "row) of parameter values.

In the case of a subsequent SQL\_UPDATE activity, the requested operation (such as insert, update or delete) specified for the activity will be performed once for each set of entries (or "row") of parameter values.

#### **OUTPUT Parameters:**

## SQL\_PARAMSCSV

This activity sets parameter values to be used subsequently by one of the following SQL database activities , using values from a specified CSV file:

SQL_CALL	Execute an SQL stored procedure
SQL_CALLQRYCSV	Query database using an SQL stored procedure to CSV
SQL_CALLQUERY	Query database using an SQL stored procedure
SQL_UPDATE	Update database using SQL

The parameter values are used in the SQL statement used by one of the above activities in place of any parameter markers (usually designated by a question mark) that are specified in the SQL statement.

Note that any parameter values that have been set will be cleared after executing any of the above activities or the SQL\_QUERY or SQL\_QUERYTOCSV activities. If you need to use the same set of parameters more than once, then you will need to repeat this activity before each activity that uses the parameters.

Refer also to the description of the SQL\_PARAMS activity, which performs a similar function but using processing sequence variables instead of a CSV file.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## **SQLFILEPATH** : Required

This parameter must specify the path and file name of the CSV file

containing the values to be used. The CSV file should contain one or more columns, up to a maximum of 25, each "row" of which supplies one set of the parameter values to be used.

A subsequent SQL\_CALL, SQL\_CALLQRYCSV or SQL\_CALLQUERY will use only one set (or "row) of parameter values.

In the case of a subsequent SQL\_UPDATE activity, the requested operation (such as insert, update or delete) specified for the activity will be performed once for each set of entries (or "row") of parameter values.

## **COLSEPARATOR** : Optional

You may specify the separator character that is used to separate the column values in each line of the CSV file. The most common form of CSV uses a comma as the separator, and that is the default value for this parameter if you do not specify it.

If a non-comma separator is specified, it should be 1 character in length and can consist of any character.

### **HEADINGSROW** : Optional

If the first row of the CSV file contains column headings, you should specify YES for this parameter. If this parameter specifies YES, then the activity will ignore the first row read. The default value is NO, which will cause the activity to process every row contained in the CSV file.

#### **OUTPUT Parameters:**

# SQL\_QUERY

This activity queries a database using an SQL SELECT statement and receives the results into processing sequence variable lists.

This activity does not support the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If you need the SQL statement to be variable (for example, selection values in a WHERE clause), then you must dynamically construct the SQL statement to include the variable values as constants.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## **SQLQUERY** : Required

This parameter must specify the SQL SELECT statement that will execute on the target database for the query. The following are examples of SQL SELECT statements that might be specified in this parameter:

select distinct bchnum from tutordh

```
select bchnum, ordnum, cusnum, orddat from tutordh where bchnum = '12345687890'
```

Since processing sequence variables are restricted to 256 characters, a single value may not be sufficiently large to hold some queries. For this reason, this parameter is defined as a variable list. This means that you can split your long SQL statement into more than one part and provide the parts in this parameter using a variable list. If you do this, then the SQL\_QUERY activity will re-assemble them into a single statement before execution.

## SQLMAXROWS : Optional

This parameter may specify a maximum number of rows to be returned. This guards against the possibility of SELECT statements that select much more data than was intended. If not specified, a default value of 100 (one hundred) is used. Remember that the SQL database activities are not intended and not usually suitable for high-throughput, high-volume database operations. If you expect that your query will return a large number of records, then you should possibly consider an alternate implementation, such as using a Transformation Map.

#### OUTPUT Parameters: SQLROWS :

Upon successful completion, this parameter will contain the actual number of rows returned by the query.

### SQLCOLUMN1 SQLCOLUMN2

•••

## SQLCOLUMN25

These output lists will contain a list of the values for the corresponding column for each row selected by the query.

You should specify the name of a variable list that will contain the values for each column used in your query, up to a maximum of 25.

You may specify fewer output lists than in your query, however, if you specify more, then a run-time error will occur, for example

*ERROR* – *Descriptor index not valid*, or *ERROR* – *The index 10 is out of range* 

# SQL\_QUERYTOCSV

This activity queries a database using an SQL SELECT statement and writes the results in CSV format into the specified file.

This activity does not support the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If you need the SQL statement to be variable (for example, selection values in a WHERE clause), then you must dynamically construct the SQL statement to include the variable values as constants.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

## **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## **SQLQUERY** : Required

This parameter must specify the SQL SELECT statement that will execute on the target database for the query. The following are examples of SQL SELECT statements that might be specified in this parameter:

select distinct bchnum from tutordh

```
select bchnum, ordnum, cusnum, orddat from tutordh where bchnum = '12345687890'
```

Since processing sequence variables are restricted to 256 characters, a single value may not be sufficiently large to hold some queries. For this reason, this parameter is defined as a variable list. This means that you can split your long SQL statement into more than one part and provide the parts in this parameter using a variable list. If you do this, then the SQL\_QUERY activity will re-assemble them into a single statement before execution.

## **SQLFILEPATH** : Required

This parameter must specify the path and file name of the CSV file to be created or appended by this activity. If the file already exists (and NO is specified or assumed for the SQLAPPEND parameter), then the file will be overwritten by this activity.

## SQLAPPEND : Optional

If the specified output file already exists, then you can specify YES for this parameter to cause the activity to append the new results to the end of the existing contents of the file. Otherwise the file is replaced with the new contents. If the specified output file does not already exist, then this parameter is ignored.

## SQLCCSID : Optional

This parameter applies only on IBM i servers. It allows you to specify the IBM i CCSID with which the output CSV file is created.

If the parameter is not specified, a default of \*DEFAULT is assumed, which instructs the activity to create the output text file using the CCSID for the job in which the activity is executing. (You can also specify the special value of \*JOB which has the same effect.)

Otherwise, you should specify the numeric CCSID value required. For example, a value of 1208 means UTF-8. Refer to IBM i documentation for a complete list and description of the available CCSIDs.

NOTE: the assumed or explicit CCSID is applied only when the specified output file does NOT already exist. If the specified output file already exists and is being replaced or appended-to by this activity, then its CCSID will not be changed.

## SQLCOLUMNS : Optional

This parameter must specify the number of columns from the expected result set that are to be written to the CSV file, up to a maximum of 25. This, of course, will depend on the SELECT statement you are using.

You may specify fewer columns than in your query, however, if you specify more, then a run-time error will occur, for example

## ERROR – Descriptor index not valid, or ERROR – The index 10 is out of range

## **COLSEPARATOR** : Optional

You may specify the separator character that is used to separate the column values in each line of the CSV output file. The most common form of CSV uses a comma as the separator, and that is the default value for this parameter if you do not specify it.

If a non-comma separator is specified, it should be 1 character in length and can consist of any character.

## **COLHEADINGS** : Optional

If you wish the first row of the output CSV file to contain comma-separated column headings, you may specify the heading line in this parameter. Note you should specify the entire string, including the separators, as a single value – for example:

### Heading1, Heading2, Heading3

If you do not specify a value for this parameter, then no heading row will be written to the CSV file. If you specified YES for the SQLAPPEND parameter, then the COLHEADINGS value is not used.

## SQLMAXROWS : Optional

This parameter may specify a maximum number of rows to be returned. This guards against the possibility of SELECT statements that select much more data than was intended.

If not specified, a default value of 100 (one hundred) is used. Remember that the SQL database activities are not intended and not usually suitable for high-throughput, high-volume database operations. If you expect that your query will return a large number of records, then you should possibly consider an alternate implementation, such as using a Transformation Map.

#### OUTPUT Parameters: SQLROWS :

Upon successful completion, this parameter will contain the actual number of rows returned by the query and written to the CSV file.

## SQL\_ROLLBACK

This activity rolls back a database transaction performed by one or more prior SQL database activities such as SQL\_UPDATE.

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### SQLHANDLE : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

#### **OUTPUT Parameters:**

## SQL\_UPDATE

This activity executes a specified SQL statement to update a database. Typically, this will be an INSERT, DELETE or UPDATE statement.

This activity supports the use of SQL parameter markers (usually designated by a question mark) in the SQL statement. If your solution uses this capability, you must first execute one of the following activities to supply the parameter value(s):

SQL\_PARAMS Set parameter values for SQL operation

SQL\_PARAMSCSV Set parameter values for SQL operation from CSV

For more information about the SQL database activities, refer to the description of the SQL\_CONNECT activity. For a list of supplied working (\*) examples using the SQL database activities, refer to:

Example Processing Sequences using the SQL database activities

### **INPUT Parameters:**

### **SQLHANDLE** : Required

This parameter must specify the connection handle value that identifies the SQL connection upon which this activity should operate. The connection handle value is returned by the SQL\_CONNECT activity.

## **SQLUPDATE** : Required

This parameter must specify the SQL statement that will execute to update the target database. Typically, this will be an INSERT, DELETE or UPDATE statement.

The SQL statement may make use of parameter markers (usually designated by a question mark). You must use either the SQL\_PARAMS or the SQL\_PARAMSCSV activity *before* this activity, to supply the parameter values.

Where the SQL\_PARAMS or the SQL\_PARAMSCSV activity have been used to supply parameter values, the SQL statement will be prepared before executing it repeatedly for each supplied set (or "row") of parameter values. The following is an example of such an SQL update statement that contains parameter markers:

update tutordh set orddat = ? where bchnum = ?

Where parameter values are *not* supplied, the statement is not prepared before it is executed and it will execute only once. The following is an example of such an SQL update statement that does NOT contain parameter markers:

update tutordh set orddat = '2014-01-01' where bchnum = '1234567890'

Since processing sequence variables are restricted to 256 characters, a single value may not be sufficiently large to hold your SQL statement. For this reason, this parameter is defined as a variable list. This means that you can split your long SQL statement into more than one part and provide the parts in this parameter using a variable list. If you do this, then the

SQL\_UPDATE activity will re-assemble them into a single statement before preparing and executing it.

### OUTPUT Parameters: SQLROWS :

If parameter values are *not* supplied for use with the SQL statement, then this parameter will contain the count of the number of database rows affected by the SQL statement.

If parameter values *are* supplied (and the statement is consequently prepared before being executed), then the activity cannot return the affected rows count and this parameter is not used.

## SUBSTITUTE

This activity formats an input string, replacing variable placeholders in the form &1, &2 ... &9 with specified values, which may be specified as literals or variables.

NB: This activity performs a very similar function to the SUBSTITUTE\_VAR activity but uses a different means of specifying the variable data to be substituted into the string. You can use either activity according to your requirements or preference.

#### **INPUT Parameters:**

### **STRINGIN** : Required

This parameter specifies the string that is to be formatted. The string may contain placeholders in the form &1, &2 ... &9 that will be replaced by the values of the corresponding variable parameters.

For example, if you specify the string "&1 files processed from &2", the placeholders &1 and &2 will be replaced by the values of the VARIABLE1 and VARIABLE2 parameters in the resulting formatted string.

#### VARIABLE1VARIABLE2 VARIABLE3 VARIABLE4 VARIABLE5 VARIABLE6 VARIABLE7 VARIABLE8 VARIABLE9 : Optional

These parameters may be used to specify the values that are to replace the variable placeholders in the format string. For example, the value of VARIABLE1 will replace each instance of &1 in the format string, while the value of VARIABLE2 will replace each instance of &2 and so on. You only need to specify values for as many variables as you have specified variable placeholders in the format string. However, they must be specified contiguously (the activity stops looking after the first parameter whose value is not specified).

#### OUTPUT Parameters: STRINGOUT :

Upon completion, this parameter will contain the formatted string.

## SUBSTITUTE\_VAR

This activity formats an input string, replacing embedded references to processing sequence variables and/or built-in variables with their current values.

NB: This activity performs a very similar function to the **SUBSTITUTE** activity but uses a different means of specifying the variable data to be substituted into the string. You can use either activity according to your requirements or preference.

#### **INPUT Parameters:**

### **STRINGIN** : Required

This parameter specifies the string that is to be formatted. The string may contain embedded references to processing sequence variables (eg: &myvar) and/or built-in variables (eg: \*tradingpartner) that will be replaced by the current variable values. Support for indexed variable references (eg: &myvar(2) OR &myvar(&myindex)) is provided.

For example, if you specify the string "&MYCOUNT files processed from &MYFOLDER(&MYINDEX) at \*now\_local", the variable and built-in variable references &MYCOUNT, &MYFOLDER(&MYINDEX) and \*now\_local will be replaced by the values of the variables that they represent.

In most cases, a reference to a processing sequence variable or builtinvariable must be followed by a space (or certain special characters or the end of the string) in order for the reference to be correctly identified and substituted.

If two ampersands (&&) or two asterisks (\*\*) appear together in the string, they are reduced to a single ampersand or asterisk and not considered for substitution.

#### OUTPUT Parameters: STRINGOUT :

Upon completion, this parameter will contain the formatted string.

## SUBSTRING

This activity extracts a portion of the input string. The portion extracted is determined by the specified starting position and length.

### **INPUT Parameters:**

**STRINGIN** : Required

This parameter specifies the string from which the substring is to be extracted.

#### **START** : Required

This parameter specifies the starting position in the source string from which to extract the substring.

**LENGTH** : Required

This parameter specifies the length of substring to be extracted.

#### **OUTPUT Parameters:** STRINGOUT :

Upon completion, this parameter will contain the extracted substring.

## SYSTEM\_COMMAND

This activity executes an operating system command. Command execution is synchronous - the processing sequence waits until the command completes before proceeding. The parameters specify the command string to be executed and the response or result code from the operating system is returned in the RESPONSE parameter.

### Notes:

\* By using this activity, you are introducing an operating system dependency into your processing sequence.

\* The specific commands you can use and their format and syntax are determined by the operating system. Refer to relevant operating system documentation.

\* The command must be eligible to be executed in the environment in which the processing sequence will run, including with respect to the required authorities.

\* If the command does not execute successfully, this is indicated by the response code. The activity ends normally. If you wish to treat the failure as an error, you must test the RESPONSE parameter.

\* You cannot call the LANSA X\_RUN entry point (to execute LANSA RDMLX applications) using this activity.

## **INPUT Parameters:**

## **COMMAND01** : Required

This parameter must contain the command to be executed. The command names and the format and syntax of the command string is operating systemdefined. Up to 256 characters may be specified. If the required command string is longer than this, you may specify the remainder of the command string in the COMMAND02 and COMMAND03 parameters.

This parameter is required. If it is not provided, the acivity will return an error.

## COMMAND02 COMMAND03 : Optional

These parameters can contain optional parts 2 and 3 of the command string. This can be used in the event that the command string is longer than the maximum supported 256 characters for the COMMAND01parameter. If present, the values of these parameters are appended to the command string before executing it. Trailing blanks are trimmed from the command string before appending these values - this means that if you need a blank space

between the command strings in parts 1 and 2 or parts 2 and 3, then the blank space must be present at the beginning of the second of the two parts.

#### OUTPUT Parameters: RESPONSE :

Upon completion, this parameter contains the response code from the operating system arising from executing the specified system command. A response code of zero indicates successful execution. On IBM i servers, a response code of 1 indicates failure. On Windows servers, one of a range of Win32 error codes may be returned in the event of an error.

## TEXT\_SUBSTITUTE

This Activity reads a skeleton text file and replaces %%parm.parmeter%% references in the text with the current parameter value and writes the expanded text file.

This Activity can be used to create a skeleton text for such things as email body text or email attachments. The complete text can then be resolved at execution time with the current execution parameter values.

Skeleton file contents example:

Thank you %%PARM.CUSNAME%% for your order. Your order, numbered %%parm.orderno%% has been dispatched via carrier %%parm.CARRIER%%.

If parameter values were:

CUSNAME	John Brown
ORDERNO	AB0015
CARRIER	FEDEX

then the resulting text file would be:

Thank you John Brown for your order. Your order, numbered AB0015 has been dispatched via carrier FEDEX.

## **INPUT Parameters:**

**TEXTSKELETON** : Required

Path and file name of the skeleton text file.

For example: c:\mydir\emailtext.txt

**TEXTFILE** : Required

Path and file name of the text file to be created.

For example: c:\mail\orderemail.txt

## TEXTCCSID : Optional

This parameter applies only on IBM i servers. It allows you to specify the IBM i CCSID with which the output text file is created.

If the parameter is not specified, a default of \*SKELETON is assumed, which instructs the activity to create the output text file using the same

CCSID as the skeleton text file.

You can also specify special values of \*DEFAULT or \*JOB which cause the activity to use the default CCSID for the job in which the activity is executing.

Otherwise, you should specify the numeric CCSID value required. For example, a value of 1208 means UTF-8. Refer to IBM i documentation for a complete list and description of the available CCSIDs.

NOTE: the assumed or explicit CCSID is applied only when the specified output file does NOT already exist. If the specified output file already exists and is being replaced by this activity, then its CCSID will not be changed.

#### **OUTPUT Parameters:**

## TRANSFORM

This activity executes a transformation map. The parameters specify the identifier (or name) of the transformation map to run and up to ten transformation map parameters.

Usually you will use the Transform directive from the palette to run a transformation map in a processing sequence. This is the easiest way when you know at design time the identity of the transformation map you wish to use because the processing sequence editor shows the exact parameters that you must provide.

However, you can execute a transformation map using this activity when the identity of the transformation map is variable. For example, you might use this to execute transformation maps that are linked to trading partner definitions. Refer to 2.4.7 Link Transformation Maps to a Trading Partner for more information.

**Note:** When you use this activity to execute a transformation map, you must ensure that the parameters specified are correct and appropriate for every different map that might be executed by the activity. You need to take care when editing the transformation maps to ensure that this is the case.

## **INPUT Parameters:**

## TRANSFORMID : Required

This parameter must specify the identity of the transformation map to be run.

# TRANSFORMP1...

#### TRANSFORMP10 : Optional

These parameters provide up to ten parameters for the transformation map. You must make sure that you specify parameters that are appropriate for every transformation map that the activity might run.

### **OUTPUT Parameters:**

## TS\_CAPTURE

The TS\_CAPTURE activity captures a representation of the current aXes Terminal Server screen and returns it in the AXCAPTURE list. The screen capture is a simple text representation. 5250 display attributes (such as underlining of input fields) are not represented.

The most common use for the TS\_CAPTURE activity is for diagnostic purposes, especially while designing a process that uses the aXes terminal server. You can use the LOGLIST activity with the list returned in the AXCAPTURE parameter to include the screen capture in the processing sequence log.

#### **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session from which the screen representation is captured. The connection handle value is returned by the TS\_CONNECT activity.

### **OUTPUT Parameters:**

#### AXSCREEN :

Upon successful completion this parameter contains the current aXes 5250 screen name or signature. The screen name is the user-defined name if an aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned.

## AXCAPTURE :

Upon successful completion, this parameter provides a list of captured aXes screen image lines. The number of lines in the list depends on the display attributes of the current 5250 screen shown in the aXes terminal server session, but is most commonly 24 or 27.

# TS\_CONNECT

The TS\_CONNECT activity is used to connect to an aXes terminal server and signon the 5250 session using the server identification and user credentials provided.

It is possible to use more than one aXes terminal server session at a time in your processing sequence. The value of the AXHANDLE output parameter identifies the session created by this instance of the TS\_CONNECT activity. The same value must be specified as the AXHANDLE input parameter value for all subsequent aXes terminal server activities (such as TS\_SETBYNAME or TS\_SEND) that are to operate on the same terminal server session.

NOTE: The processing sequence containing the TS\_CONNECT activity becomes ineligible to be restarted in the event of any failure for the duration of the aXes terminal server session - that is, until the TS\_DISCONNECT activity is executed.

### **INPUT Parameters:**

### **HOST** : Required

This parameter must specify the host name or IP address of the machine hosting the aXes terminal server to which you wish to connect.

## **PORT** : Optional

This parameter may specify the port number to be used to connect to the aXes terminal server. If not specified, a default value of 80 is assumed.

### **USER** : Required

This keyword must specify the user name used to connect to the aXes terminal server. This must be the name of an authorised i/OS user profile.

## **PASSWORD** : Required

This parameter must specify the password for the user name specified in the USER keyword.

## AXPROJECT : Optional

This parameter may be used to set the aXes project to use for this session. A project contains user assigned definitions of screens and fields. For example, you can assign a name to a screen or a field, and later use that name to refer to the screen or the field in other aXes terminal server activities. Although aXes projects can either be stored in a file on the file system or on the aXes server, you may only specify here an aXes server-based project.

## AXEXTENDEDTRACE : Optional

If you specify YES for this parameter, LANSA Composer will use additional LANSA Integrator aXesTerminalService tracing options that will create additional Java Service Manager (JSM) tracing. The additional tracing will include terminal session screen captures at appropriate points as well as creating an additional aXes-specific trace file in the JSM trace directory. The additional tracing may help in diagnosing interactions with the aXes terminal server and so may be useful in the design and testing stage of your LANSA Composer solution. However, there is additional processing overhead associated with the the extra tracing and you may wish NOT to use it in a performance-sensitive production environment. The effect of this parameter is subject to standard LANSA Composer logging and tracing system settings being in effect. The default for this parameter, if not specified, is NO.

### **OUTPUT Parameters:**

### **AXHANDLE** : Required

If successful, the value of this output parameter identifies the session created by this instance of the TS\_CONNECT activity. The same value must be specified as the AXHANDLE input parameter value for all subsequent aXes terminal server activities that are to operate on the same terminal server session.

## AXSCREEN :

Upon successful completion this parameter contains the resulting aXes 5250 screen name or signature. The screen name is the user-defined name if an aXes project name was specified in the AXPROJECT parameter and a name was assigned to the screen. Otherwise the screen signature is returned.

## **TS\_DISCONNECT**

The TS\_DISCONNECT activity disconnects the specified aXes terminal server session.

### **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session that should be disconnected. The connection handle value is returned by the TS\_CONNECT activity.

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

## TS\_EXECUTE

The TS\_EXECUTE activity executes a specified routine in an aXes terminal operation script. Refer to the LANSA Composer guide for more information about aXes terminal operations scripts.

### **INPUT Parameters:**

### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session upon which the aXes terminal operations script should operate. The connection handle value is returned by the TS\_CONNECT activity.

### AXSCRIPTFILE : Required

This parameter must specify the full path to a file that contains the aXes terminal operations script routine to be executed. The aXes terminal operations script must be in a comma-separated format as described in the LANSA Composer guide.

#### AXROUTINE : Optional

This parameter may specify the name of a routine in the aXes terminal operation script. If not specified, a blank routine name is assumed. If your script does not contain named routines you do not need to specify this parameter.

### AXRELOAD : Optional

You may specify YES in this parameter to force the aXes terminal operations script to be re-loaded from the file specified in the AXSCRIPTFILE parameter. In normal operation, the activity loads an aXes terminal operations script file once and then uses the loaded copy if the same script is specified again in subsequent TS\_EXECUTE activity executions. However, if your processing sequence "generates" the script file in response to other inputs (by using a Transformation Map, for example), then you may need to force the TS\_EXECUTE activity to reload the script after your processing sequence has changed it. In that case you should specify YES for this parameter. If not specified, the default for this parameter is NO.

#### **OUTPUT Parameters:** AXSCREEN :

Upon successful completion this parameter contains the resulting aXes 5250 screen name or signature. The screen name is the user-defined name if an

aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned.

# TS\_GET

The TS\_GET activity retrieves attributes associated with the specified aXes terminal session.

### **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session whose attributes are to be retrieved. The connection handle value is returned by the TS\_CONNECT activity.

#### OUTPUT Parameters: AXCURSORROWOUT :

If successful, this parameter will contain the row number component of the current screen cursor position.

## **AXCURSORCOLOUT** :

If successful, this parameter will contain the column number component of the current screen cursor position.

## AXSYSTEM :

If successful, this parameter will contain the i/OS system name for the current aXes terminal session.

## **AXDEVICE** :

If successful, this parameter will contain the 5250 device name used by the current aXes terminal session.

## **AXJOBUSER** :

If successful, this parameter will contain the user name portion of the i/OS job name for the current aXes terminal session.

## **AXJOBNUMBER** :

If successful, this parameter will contain the job number portion of the i/OS job name for the current aXes terminal session.

## **AXSCREENWIDTH** :

If successful, this parameter will contain the current screen width for the aXes terminal session. Note that this value refers to the screen currently displayed, not maximum device capabilities.

## **AXSCREENHEIGHT** :

If successful, this parameter will contain the current screen height for the

aXes terminal session. Note that this value refers to the screen currently displayed, not maximum device capabilities.

## AXSCREEN :

Upon successful completion this parameter contains the current aXes 5250 screen name or signature. The screen name is the user-defined name if an aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned.
# TS\_GETBYNAME

The TS\_GETBYNAME activity retrieves the value of a field on the current aXes terminal screen by reference to the field name (and an optional index for a subfile field).

#### **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session from which the screen field value is to be retrieved. The connection handle value is returned by the TS\_CONNECT activity.

#### **AXNAME** : Required

This parameter must specify the name of the field whose value is to be retrieved. The name may be a user-defined name (if an aXes project name was specified on the TS\_CONNECT activity) or an aXes field identifier.

#### AXINDEX : Optional

If the field is contained in a subfile, this parameter may specify the index of the instance on the current screen of the field whose value is to be retrieved. The first instance on the current screen has an index of 1, irrespective of the scroll position of the subfile. If your processing sequence needs to process fields in subfiles, it must do so a screen at a time and send ROLL\_UP/ROLL\_DOWN using the TS\_SEND activity (just like a 5250 terminal user would have to do).

#### **OUTPUT Parameters:**

#### AXSCREENVALUE :

Upon successful completion, this parameter will contain the value for the specified screen field.

# TS\_GETBYPOS

The TS\_GETBYPOS activity retrieves the value of a field on the current aXes terminal screen by reference to the screen row and column number.

## **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session from which the screen field value is to be retrieved. The connection handle value is returned by the TS\_CONNECT activity.

#### AXSCREENROW : Required

This parameter must specify the screen row number of the required field. The row and column together must refer to the first position of the required field.

#### AXSCREENCOL : Required

This parameter must specify the screen column number of the required field. The row and column together must refer to the first position of the required field.

#### OUTPUT Parameters: AXSCREENVALUE :

Upon successful completion, this parameter will contain the value for the specified screen field.

# TS\_GETFIELD

The TS\_GETFIELD activity retrieves the attributes of a field on the current aXes terminal screen by reference to either the field name (and an optional index for a subfile field) or to the screen row and column number of the field.

### **INPUT Parameters:**

## **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session from which the screen field attributes are to be retrieved. The connection handle value is returned by the TS\_CONNECT activity.

## AXNAME : Optional

This parameter may specify the name of the field whose attributes are to be retrieved. The name may be a user-defined name (if an aXes project name was specified on the TS\_CONNECT activity) or an aXes field identifier. If this parameter is not specified, then values must be specified for the AXSCREENROW and AXSCREENCOL parameters.

## AXINDEX : Optional

If a field identified by name (the AXNAME parameter) is contained in a subfile, this parameter may specify the index of the instance on the current screen of the field whose attributes are to be retrieved. The first instance on the current screen has an index of 1, irrespective of the scroll position of the subfile. If your processing sequence needs to process fields in subfiles, it must do so a screen at a time and send ROLL\_UP/ROLL\_DOWN using the TS\_SEND activity (just like a 5250 terminal user would have to do).

## AXSCREENROW : Optional

This parameter may specify the screen row number of the required field. The row and column together, if used, must refer to the first position of the required field. The value of this keyword is ignored if a value is specified for the AXNAME parameter. Conversely, if AXNAME is not specified, then AXSCREENROW and AXSCREENCOL parameter values must be supplied.

## AXSCREENCOL : Optional

This parameter may specify the screen column number of the required field. The row and column together, if used, must refer to the first position of the required field. The value of this keyword is ignored if a value is specified for the AXNAME parameter. Conversely, if AXNAME is not specified, then AXSCREENROW and AXSCREENCOL parameter values must be supplied.

## **OUTPUT Parameters:**

# AXNAMEOUT :

If successful, this parameter will contain the user-defined name of the field (if an aXes project name was specified on the CONNECT command) or the aXes field identifier if no project was specified or no name has been defined for the field.

# **AXINDEXOUT** :

If successful, this parameter will contain the index of the instance on the current screen of the field. The first instance on the current screen has an index of 1, irrespective of the scroll position of the subfile.

## AXSCREENROWOUT :

If successful, this parameter will contain the screen row number of the field.

# AXSCREENCOLOUT :

If successful, this parameter will contain the screen column number of the field.

## **AXSCREENCOUNT** :

If successful and the specified field is contained in a subfile, this parameter will contain the number of instances on the current screen for the field (the number of subfile records displayed on the current screen). For reliability, you should use this activity for a field that is never conditioned on the subfile display - such as the option entry field in a "Work with ..." style display.

## AXSCREENVALUE :

If successful, this parameter will contain the value for the specified screen field.

# TS\_SEND

The TS\_SEND activity sends the current aXes screen data to the aXes terminal server. It is analogous to pressing the ENTER key (by default) or a function key at a 5250 terminal.

## **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session upon which this activity should operate. The connection handle value is returned by the TS\_CONNECT activity.

#### AXKEY : Optional

This parameter may specify the function key to send. You may use any of the values listed below. If not specified, \*ENTER is sent.

\*ENTER \*F1 - \*F9 \*F10 - \*F24 \*PAGE\_UP, \*PAGE\_DOWN \*ROLL\_UP, \*ROLL\_DOWN \*HELP, \*PRINT

## AXCURSORROW : Optional

This parameter may specify the screen cursor row number to send. If AXCURSORROW and AXCURSORCOL are not both specified, the screen cursor position is unchanged.

#### AXCURSORCOL : Optional

This parameter may specify the screen cursor column number to send. If AXCURSORROW and AXCURSORCOL are not both specified, the screen cursor position is unchanged.

#### **OUTPUT Parameters:** AXSCREEN :

Upon successful completion this parameter contains the resulting aXes 5250 screen name or signature. The screen name is the user-defined name if an aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned.

# TS\_SETBYNAME

The TS\_SETBYNAME activity sets the value of a field on the current aXes terminal screen by reference to the field name (and an optional index for a subfile field) and may (optionally) send the current aXes screen data to the aXes terminal server. It is analogous to typing at a 5250 terminal.

## **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session upon which this activity should operate. The connection handle value is returned by the TS\_CONNECT activity.

#### **AXNAME** : Required

This parameter must specify the name of the field whose value is to be set. The name may be a user-defined name (if an aXes project name was specified on the TS\_CONNECT activity) or an aXes field identifier.

#### AXINDEX : Optional

If the field is contained in a subfile, this parameter may specify the index of the instance on the current screen of the field whose value is to be set. The first instance on the current screen has an index of 1, irrespective of the scroll position of the subfile. If your processing sequence needs to process fields in subfiles, it must do so a screen at a time and send ROLL\_UP/ROLL\_DOWN using the TS\_SEND activity (just like a 5250

terminal user would have to do).

## AXSCREENVALUE : Required

This parameter must specify the value to be set for the specified screen field.

#### AXKEY : Optional

This parameter may optionally specify a function key to send the current aXes screen data to the aXes terminal server. You may use any of the values listed below. If not specified, no send is performed.

\*ENTER \*F1 - \*F9 \*F10 - \*F24 \*PAGE\_UP, \*PAGE\_DOWN \*ROLL\_UP, \*ROLL\_DOWN \*HELP, \*PRINT

#### OUTPUT Parameters: AXSCREEN :

Upon successful completion this parameter contains the resulting aXes 5250 screen name or signature. The screen name is the user-defined name if an aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned. (NB: The 5250 screen that is presently shown can be affected or changed by executing this activity ONLY if a function key value is specified in the AXKEY parameter.)

# TS\_SETBYPOS

The TS\_SETBYPOS command sets the value of a field on the current aXes terminal screen by reference to the screen row and column number and may (optionally) send the current aXes screen data to the aXes terminal server. It is analogous to typing at a 5250 terminal.

## **INPUT Parameters:**

## **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session upon which this activity should operate. The connection handle value is returned by the TS\_CONNECT activity.

## AXSCREENROW : Required

This parameter must specify the screen row number of the required field. The row and column together must refer to the first position of the required field.

#### AXSCREENCOL : Required

This parameter must specify the screen column number of the required field. The row and column together must refer to the first position of the required field.

#### AXSCREENVALUE : Required

This parameter must specify the value to be set for the specified screen field.

## AXKEY : Optional

This parameter may optionally specify a function key to send the current aXes screen data to the aXes terminal server. You may use any of the values listed below. If not specified, no send is performed.

```
*ENTER
*F1 - *F9
*F10 - *F24
*PAGE_UP, *PAGE_DOWN
*ROLL_UP, *ROLL_DOWN
*HELP, *PRINT
```

#### **OUTPUT Parameters:** AXSCREEN :

Upon successful completion this parameter contains the resulting aXes 5250

screen name or signature. The screen name is the user-defined name if an aXes project name was specified on the TS\_CONNECT activity and a name was assigned to the screen. Otherwise the screen signature is returned. (NB: The 5250 screen that is presently shown can be affected or changed by executing this activity ONLY if a function key value is specified in the AXKEY parameter.)

# TS\_SETCURSOR

The TS\_SETCURSOR activity sets the position of the cursor on the current aXes terminal screen. It is analogous to using the arrow or tab keys to move the cursor at a 5250 terminal. Some 5250 applications or screens are sensitive to the cursor position and may exhibit different functionality dependent upon it.

## **INPUT Parameters:**

#### **AXHANDLE** : Required

This parameter must specify the connection handle value that identifies the aXes terminal server session upon which this activity should operate. The connection handle value is returned by the TS\_CONNECT activity.

#### AXCURSORROW : Required

This parameter must specify the cursor row number to set.

#### AXCURSORCOL : Required

This parameter must specify the cursor column number to set.

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

# TXDOC\_ALLOCCTRL

This activity allocates one or more control numbers for outbound transaction documents. It can allocate:

- zero, one or more interchange control numbers
- zero, one or more group control numbers
- zero, one or more transaction set (message) control numbers

The control number(s) are allocated from the series that apply to the specified trading partner, as can be seen on the *Outbound Numbering* tab of the Trading Partner definition, and the "next" numbers for the series are incremented accordingly.

If the outbound numbering domain for the trading partner specifies that outbound control numbers are allocated by transaction id, then the DOCCONTENTTYPE parameter must specify the transaction type of the series from which the control numbers are to be allocated.

NOTE (1) - When using the API supplied with LANSA Composer for registering pending outbound documents, the necessary interchange, group and transaction set control numbers are automatically allocated by LANSA Composer. You do not need to use this activity in that event. The same usually applies when using an export processor associated with a transformation map for an outbound transaction document process (although this will depend on the specific implementation of the export processor).

NOTE (2) - There is no means to de-allocate or free control numbers once allocated. If your trading environment or trading agreements requires sequential control numbering or auditability of allocated control numbers, then it is your responsibility to satisfy those requirements when using this activity to allocate control numbers.

## **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the trading partner for which the control numbers are allocated. This parameter is required.

## **DOCCONTENTTYPE** : Optional

This parameter may be used to specify the document content type for which

the control numbers are allocated. It is required if the outbound numbering domain for the trading partner specifies that outbound control numbers are allocated by transaction id. However, if the outbound numbering domain for the trading partner specifies that outbound control numbers are allocated across all transaction ids, then the value of the DOCCONTENTTYPE parameter will be ignored.

## ALLOCATE\_IC : Optional

Specifies how many interchange control numbers to allocate. The allocated interchange control numbers will be placed in the CTRLNUMBER\_IC output list parameter.

## ALLOCATE\_GP : Optional

Specifies how many group control numbers to allocate. The allocated group control numbers will be placed in the CTRLNUMBER\_GP output list parameter.

#### ALLOCATE\_MS : Optional

Specifies how many transaction set (message) control numbers to allocate. The allocated transaction set control numbers will be placed in the CTRLNUMBER\_MS output list parameter.

#### **OUTPUT Parameters:**

#### CTRLNUMBER\_IC :

Upon successful completion, this parameter will contain a list of zero, one or more allocated interchange control numbers, according to the value specified in the ALLOCATE\_IC parameter.

#### **CTRLNUMBER\_GP**:

Upon successful completion, this parameter will contain a list of zero, one or more allocated group control numbers, according to the value specified in the ALLOCATE\_GP parameter.

#### **CTRLNUMBER\_MS** :

Upon successful completion, this parameter will contain a list of zero, one or more allocated transaction set (message) control numbers, according to the value specified in the ALLOCATE\_MS parameter.

# TXDOC\_EXPORT

This activity exports transaction data for out-going transactions by executing the re-useable part specified as the export processor (the export processor is typically associated with the transformation map that will be used to transform the data to its out-going transaction document format).

Refer to the supplied processing sequence TXDOC\_IEDI090 as an example of using this activity in the course of generating an outbound EDI X12 997 acknowledgement in response to an in-coming EDI document.

#### **INPUT Parameters:**

#### DOCEXPORTPROCESSOR : Required

This parameter specifies the name of the transaction data export processor component name. This must be the name of a re-useable part derived from the supplied ancestor class DXXEXPBAS. This component must be written to be capable of performing the required processing to export the transaction data identified by the transaction data identifiers, typically into staging files that will subsequently be used by the associated transformation map to create an outbound transaction document file containing the data.

### **DOCPRODTEST** : Optional

This parameter specifies whether the transaction document will be for production (P) or test (T) use. It is up to the specific export processor implementation how it makes use of this value - for EDI X12 transactions, it would typically be used to fill the ISA15 (use) field of the interchange header.

#### DOCDATAKEY1DOCDATAKEY2 : Optional

These parameters are used to identify to the export processor the set of transaction data to be exported. For example, they may be used to identify a sales order, an invoice or another transaction document envelope to which this is generating a response or acknowledgement. The meaning and interpretation of these values is up to the implementation of the particular export processor being used. The values supplied must be appropriate for the use in that export processor.

#### OUTPUT Parameters: DOCNUMBERLIST :

Upon successful completion, this will contain a list of one or more transaction document envelope numbers generated by the activity.

# TXDOC\_IMPORT

This activity imports transaction data for incoming transactions by executing the re-useable part specified as the import processor (the import processor is typically associated with the transformation map used to transform the incoming data).

Refer to the supplied processing sequence TXDOC\_IEDI020 as an example of using this activity in the course of processing an in-coming EDI document.

#### **INPUT Parameters:**

#### **DOCNUMBER** : Required

This parameter must specify the transaction document envelope number for the transaction document whose data is being imported. This number is normally assigned by the TXDOC\_REGISTER activity.

## **DOCCONTENTTYPE** : Required

This parameter specifies the document content type for the transaction document. The content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: '850') for an EDI document or the XML document root element name for an XML document.

## **DOCIMPORTPROCESSOR** : Required

This parameter specifies the name of the transaction data import processor component name. This must be the name of a re-useable part derived from the supplied ancestor class DXXIMPBAS. This component must be written to be capable of performing the required processing to import the transaction data for this transaction document, typically from staging files that have been populated by the associated transformation map.

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

# TXDOC\_KEYS

This activity retrieves the control numbers and staging file key values for each transaction set (message) applicable to the specified pending outbound transaction document.

This can optionally be used as an alternate means, for example, of passing the staging file key values to the outbound transformation map.

Refer also to the description of the FOR\_EACH\_TXDOCT activity which performs a similar function, but implemented as an iterator activity.

## **INPUT Parameters:**

## **DOCNUMBER** : Required

Specifies the transaction document envelope number whose control numbers and staging file key values are to be retrieved.

# **OUTPUT Parameters:**

## CTRLNUMBER\_IC :

Upon successful completion, this parameter will contain a list of the interchange control numbers for each transaction set (message) registered for the document.

## **CTRLNUMBER\_GP**:

Upon successful completion, this parameter will contain a list of the group control numbers for each transaction set (message) registered for the document.

## **CTRLNUMBER\_MS** :

Upon successful completion, this parameter will contain a list of the transaction set (message) numbers for each transaction set (message) registered for the document.

## **DOCNUMBER\_IC** :

Upon successful completion, this parameter will contain a list of the internal document interchange sequence numbers for each transaction set (message) registered for the document. These values correspond to the key field DXXINTID in the staging database files DXX2IN, DXX3GP and DXX4MS.

## **DOCNUMBER\_GP**:

Upon successful completion, this parameter will contain a list of the internal document group sequence numbers for each transaction set (message) registered for the document. These values correspond to the key field

DXXGRPID in the staging database files DXX3GP and DXX4MS.

# **DOCNUMBER\_MS** :

Upon successful completion, this parameter will contain a list of the internal document transaction set (message) sequence numbers for each transaction set (message) registered for the document. These values correspond to the key field DXXMSGID in the staging database file DXX4MS.

#### DOCDATAKEY01DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 :

Upon successful completion, these parameters will contain lists of the "application-defined" staging file key values for each transaction set (message) registered for the document.

# TXDOC\_REGISTER

This activity registers an in-coming or out-going transaction document in the transaction document register. Normally this should be done as soon as the document becomes available to the process and before significant processing is done for it.

After registering the document, and as the processing sequence proceeds to process the transaction document, it records its progress and updates the status in the transaction document register using the TXDOC\_STATUS activity.

#### **INPUT Parameters:**

#### TRADINGPARTNER : Optional

Specifies the identifier of the trading partner for which the transaction document is being registered. If not specified, the activity will use the value of the \*TRADINGPARTNER built-in variable. Otherwise, you should specify a variable or literal that contains the required trading partner ID. The trading partner is required by this activity, so if you do not provide a valid trading partner identifier and the \*TRADINGPARTNER built-in variable is not set, the activity will end in error.

### **DOCDIR** : Optional

Specifies the direction (I/O) for the transaction document exchange.

#### **DOCFILE** : Required

This parameter specifies the path and name of the transaction document file.

#### **DOCTYPE** : Optional

This parameter may be used to specify the document type for the transaction document. If (as is frequently the case) the document type is not known at the time the transaction document is registered, it can be specified at a later point using the TXDOC\_STATUS activity. If specified, the document type should be a document type identifier as defined in LANSA Composer's *Document types* definitions, for example 'EDI'.

## **DOCCONTENTTYPE** : Optional

This parameter may be used to specify the document content type for the transaction document. If (as is frequently the case) the document content type is not known at the time the transaction document is registered, it can be specified at a later point using the TXDOC\_STATUS activity. If specified, the content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: '850') for an EDI

document or the XML document root element name for an XML document. **DOCPARENT** : Optional

If applicable, this parameter can specify an existing registered transaction document number that represents the parent of this transaction document. This is typically applied in cases where an in-coming compound transaction document is split into individual transaction documents before further processing.

## **DOCRESPONSETO** : Optional

If applicable, this parameter can specify an existing registered transaction document number to which this transaction document is a response. For example, when generating an acknowledgement for an in-coming transaction.

## DOCNUMBERPREASSIGNED : Optional

If a transaction document envelope number has already been assigned to the new transaction document (by some other process or activity), then it should be specified in this parameter. Otherwise the activity will assign a new transaction document envelope number.

#### **OUTPUT Parameters: DOCNUMBER** :

Upon successful completion, this parameter will contain the transaction document envelope number assigned to the new transaction document. This number should be saved and specified on subsequent TXDOC\_STATUS activity calls relating to this transaction document.

# TXDOC\_REGOUTBND

This activity registers a pending outbound transaction document for later processing by an outbound process such as the supplied TXDOC\_OUTBOUND process.

Note that this registers the necessity for the outbound document - the application typically uses this activity (or one of the functionally equivalent APIs) upon an event occurring that will require an outbound transaction document to be generated.

To put this another way, the outbound document is typically not existing when this activity is run, but, rather, it is created by the subsequent outbound transaction document process. (The activity FOR\_EACH\_TXDOCO is provided for use in the outbound transaction document process for identifying the pending outbound transaction documents).

## **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the trading partner for which the pending outbound transaction document is registered. This parameter is required.

## **DOCTYPE** : Required

This parameter specifies the document type for the pending outbound transaction document. The document type must be a document type identifier as defined in LANSA Composer's Document types definitions, for example 'EDI'.

## **DOCSTD** : Required

This parameter specifies the transaction document standard for the pending outbound transaction document. The document standard must be a document standard identifier as defined in LANSA Composer's Document standards definitions, for example 'X12'.

## **DOCSTDVER** : Required

This parameter specifies the transaction document standard version for the pending outbound transaction document. The document standard version must be a document standard version identifier as defined in LANSA Composer's Document standards definitions, for example '004020'.

## **DOCPRODTEST** : Optional

This parameter specifies whether the transaction document will be for production (P) or test (T) use. It is up to the specific export processor

implementation how it makes use of this value - for EDI X12 transactions, it would typically be used to fill the ISA15 (use) field of the interchange header.

## **DOCCONTENTTYPE** : Required

This parameter specifies the document content type for the pending outbound transaction document. The content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: '850') for an EDI document or the XML document root element name for an XML document.

## **DOCDATAKEY01** : Required

This parameter specifies application-defined "key" information that will allow the export processor to identify the application data associated with the pending outbound transaction document.

#### DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 : Optional

These parameters may be used to specify further application-defined "key" information.

#### OUTPUT Parameters: DOCNUMBER :

Upon successful completion, this parameter will contain the transaction document envelope number assigned to the new pending outbound transaction document.

# TXDOC\_REGOUTEDI

This activity is a specialised extension of the TXDOC\_REGOUTBND activity that provides additional parameters to facilitate registering pending outbound EDIFACT transaction documents. It registers the pending outbound transaction document for later processing by an outbound process such as the supplied TXDOC\_OUTBOUND process.

Note that this registers the necessity for the outbound document - the application typically uses this activity (or one of the functionally equivalent APIs) upon an event occurring that will require an outbound transaction document to be generated.

To put this another way, the outbound document is typically not existing when this activity is run, but, rather, it is created by the subsequent outbound transaction document process. (The activity FOR\_EACH\_TXDOCO is provided for use in the outbound transaction document process for identifying the pending outbound transaction documents).

#### **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the trading partner for which the pending outbound transaction document is registered. This parameter is required.

## **DOCTYPE** : Required

This parameter specifies the document type for the pending outbound transaction document. The document type must be a document type identifier as defined in LANSA Composer's Document types definitions, for example 'EDI'.

#### **DOCSTD** : Required

This parameter specifies the transaction document standard for the pending outbound transaction document. The document standard must be a document standard identifier as defined in LANSA Composer's Document standards definitions, for example 'EDIFACT'.

## **DOCSTDVER** : Required

This parameter specifies the transaction document standard version for the pending outbound transaction document. The document standard version must be a document standard version identifier as defined in LANSA Composer's Document standards definitions, for example 'D06A'.

## **DOCPRODTEST** : Optional

This parameter specifies whether the transaction document will be for production (P) or test (T) use. It is up to the specific export processor implementation how it makes use of this value - for EDIFACT transactions, it would typically be used to populate and/or condition the UNB11 (test indicator) field of the interchange header.

# **DOCCONTENTTYPE** : Required

This parameter specifies the document content type for the pending outbound transaction document. The content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: 'INVOIC') for an EDI document or the XML document root element name for an XML document.

## **DOCGROUP** : Optional

This parameter specifies whether a functional group (UNG / UNE segments) is to be included in the outbound transaction document. It defaults to 'N' (no).

Although the activity records this value in LANSA Composer's document register (in field DXXGPABS in file DXX3GP), it is your responsibility to design your outbound Transformation Map to read and respect the value in order to include or exclude the functional group from the resulting EDI document.

(If a single EDIFACT document includes messages of different types, each different type is placed into a group introduced by a UNG Functional Group Header segment and terminated by a UNE Functional Group Trailer segment. In practice, there is no need in most EDIFACT applications to mix different message types in a single transmission, and the Group level is not generally used. Specifically, it is never used in book trade and library applications.)

DOCCTY DOCISA11 DOCISA12 DOCISA14 DOCGS01 DOCTRVR DOCTRRL DOCTRCA DOCTRAC : Optional

These parameters may be used to specify values for selected EDI interchange header (UNB), group header (UNG) and message segments. These values, if

specified, will be written to LANSA Composer's document register in the files DXX2IN, DXX3GP and DXX4MS and can subsequently be accessed in the Transformation Map for the outbound document to write the values directly to the resulting EDI document. It is not mandatory that you provide all these values - depending on the particulars of your implementation, you may derive some or all of these values in an export processor and/or outbound Transformation Map before or in the course of preparing the outbound EDI document.

#### DOCDATAKEY01 DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 : Optional

These parameters may be used to specify application-defined "key" information that will allow the export processor and/or outbound Transformation Map to identify the application data associated with the pending outbound transaction document.

#### OUTPUT Parameters: DOCNUMBER :

Upon successful completion, this parameter will contain the transaction document envelope number assigned to the new pending outbound transaction document.

# TXDOC\_REGOUTX12

This activity is a specialised extension of the TXDOC\_REGOUTBND activity that provides additional parameters to facilitate registering pending outbound EDI X12 transaction documents. It registers the pending outbound transaction document for later processing by an outbound process such as the supplied TXDOC\_OUTBOUND process.

Note that this registers the necessity for the outbound document - the application typically uses this activity (or one of the functionally equivalent APIs) upon an event occurring that will require an outbound transaction document to be generated.

To put this another way, the outbound document is typically not existing when this activity is run, but, rather, it is created by the subsequent outbound transaction document process. (The activity FOR\_EACH\_TXDOCO is provided for use in the outbound transaction document process for identifying the pending outbound transaction documents).

## **INPUT Parameters:**

## TRADINGPARTNER : Required

Specifies the trading partner for which the pending outbound transaction document is registered. This parameter is required.

## **DOCTYPE** : Required

This parameter specifies the document type for the pending outbound transaction document. The document type must be a document type identifier as defined in LANSA Composer's Document types definitions, for example 'EDI'.

## **DOCSTD** : Required

This parameter specifies the transaction document standard for the pending outbound transaction document. The document standard must be a document standard identifier as defined in LANSA Composer's Document standards definitions, for example 'X12'.

## **DOCSTDVER** : Required

This parameter specifies the transaction document standard version for the pending outbound transaction document. The document standard version must be a document standard version identifier as defined in LANSA Composer's Document standards definitions, for example '004020'.

## **DOCPRODTEST** : Optional

This parameter specifies whether the transaction document will be for production (P) or test (T) use. It is up to the specific export processor implementation how it makes use of this value - for EDI X12 transactions, it would typically be used to fill the ISA15 (use) field of the interchange header.

## **DOCCONTENTTYPE** : Required

This parameter specifies the document content type for the pending outbound transaction document. The content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: '850') for an EDI document or the XML document root element name for an XML document.

DOCCTY DOCISA11 DOCISA12 DOCISA14 DOCGS01 DOCISA01 DOCISA02 DOCISA03 DOCISA04 : Optional

These parameters may be used to specify values for selected EDI X12 interchange header (ISA) and group header (GS) segments. These values, if specified, will be written to LANSA Composer's document register in the files DXX2IN and DXX3GP and can subsequently be accessed in the Transformation Map for the outbound document to write the values directly to the resulting EDI document. It is not mandatory that you provide all these values - depending on the particulars of your implementation, you may derive some or all of these values in an export processor and/or outbound Transformation Map before or in the course of preparing the outbound EDI document.

DOCDATAKEY01 DOCDATAKEY02 DOCDATAKEY03 DOCDATAKEY04 DOCDATAKEY05 DOCDATAKEY06 : Optional

These parameters may be used to specify application-defined "key" information that will allow the export processor and/or outbound

Transformation Map to identify the application data associated with the pending outbound transaction document.

#### OUTPUT Parameters: DOCNUMBER :

Upon successful completion, this parameter will contain the transaction document envelope number assigned to the new pending outbound transaction document.

# TXDOC\_STATUS

This activity updates the status of a transaction document in the transaction document register and records its progress in the transaction document log.

#### **INPUT Parameters:**

#### **DOCNUMBER** : Required

This parameter must specify the transaction document envelope number for the transaction document whose status is being updated. This number is normally assigned by the TXDOC\_REGISTER activity.

#### **DOCSTATUS** : Required

Specifies the new transaction document status. It can be specified either as one of a range of pre-defined values (listed below), OR as a user-specified status string value.

#### **DOCFILE** : Optional

This parameter may be used to specify a new path and name for the transaction document file. This is typically used when the transaction document is moved and/or renamed in the course of processing.

#### **DOCTYPE** : Optional

This parameter may be used to specify a new document type for the transaction document. This is typically used after "discovering" the document type during the course of processing. The document type should be a document type identifier as defined in LANSA Composer's *Document types* definitions, for example 'EDI'.

## **DOCCONTENTTYPE** : Optional

This parameter may be used to specify a new document content type for the transaction document. This is typically used after "discovering" the document content type during the course of processing. If specified, the content type can be any string that identifies the content type - for example, you might use the EDI transaction type (eg: '850') for an EDI document or the XML document root element name for an XML document.

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

#### **Pre-defined Transaction Document Status Values**

The following is a list of pre-defined transaction document status code values that may be specified in the DOCSTATUS parameter of the TXDOC\_STATUS

activity:

Status code	Meaning
*REGISTERED	Registered transaction document
*DOCUMENTTYPEOK	Discovered document type
*DOCUMENTTYPEUNK	Unrecognised document type
*DOCSPLITOK	Document split successfully
*SPLITDOCSOK	Split dpocuments processed successfully
*CONTENTMATCHTPOK	Matched trading partner to document content
*CONTENTTYPEOK	Discovered content type
*CONTENTTYPEUNK	Unrecognised content type
*CONTENTVALID	Validated document content successfully
*CONTENTINVALID	Invalid document content
*TRANSFORMFOUNDOK	Found transformation map for document
*TRANSFORMNOTFOUND	No matching transformation map for document
*TRANSFORMEDOK	Transformation successful for document
*TRANSFORMFAILED	Transformation failed for document
*IMPORTEDOK	Imported transaction data successfully for transaction document
*IMPORTFAILED	Import failed for transaction document
*ACKNOWLEDGEDOK	Acknowledgement completed for transaction document
*ACKNOWLEDGEFAILED	Acknowledgement failed for transaction document
*READYTOSENDOK	Ready to send transaction document
*COMPLETEOK	Successfully completed processing transaction document

\*COMPLETEERROR

Error(s) occurred while processing transaction document

# UNIQUEID

This activity generates a unique (\*) identifier. The identifier might be suitable for use as a database surrogate key, file name, message identifier or for many other purposes.

The generated identifier is a thirty-two character string consisting entirely of digits 0-9 and characters A-F. For example:

31EC378D30074175B15212E11ACEAF0C.

(\*) For almost all purposes and circumstances the generated identifier will be unique in space and in time - that is across all other instances generated by this activity at any time on the same or different servers.

#### **INPUT Parameters:**

None.

#### OUTPUT Parameters: UNIQUEID :

Upon completion, this parameter contains the generated 32 character identifier.

# UPPERCASE

This activity returns the specified input string with all lowercase characters converted to uppercase.

#### **INPUT Parameters:**

**STRINGIN** : Required

This parameter specifies the string to be converted.

#### **OUTPUT Parameters: STRINGOUT** :

Upon completion, this parameter will contain the converted string.

# WAIT\_FILESREADY

This activity is intended for use in processing sequences that identify and process newly-discovered files. For example, a processing sequence that uses the WATCH\_DIRECTORY activity may need to also use this activity.

Such applications may not be able to safely assume that the file just discovered is immediately available for further processing. On the contrary, the detection of a new file will frequently precede the file's availability for further processing – often because the application that created the file is still writing its contents. The WAIT\_FILESREADY activity provides services designed to allow LANSA Composer solutions to address these issues and prevent contention conditions

that may otherwise arise. These services include:

- Providing a cache of files awaiting "ready" status;
- Permitting the solution to add newly-discovered files to the cache;
- Performing periodic checks on the status and selected metrics of each cached file to determine its "ready" status;
- Notifying the solution when a cached file is considered "ready" for further processing.
- The file cache that the WAIT\_FILESREADY activity provides is "global" to the Processing Sequence. In other words, multiple instances of the WAIT\_FILESREADY activity in one processing sequence will refer to the same cache.

This is a specialized and advanced activity that behaves in a different way to many other LANSA Composer activities that you may be more used to. You should ensure you thoroughly understand the activity before designing your solution to use it.

This activity will usually appear in some sort of loop in a processing sequence – for example inside the block associated with an instance of the WATCH\_DIRECTORY activity:

- When executed inside the loop, the processing sequence usually wants to add a newly-discovered file to the cache provided by the WAIT\_FILESREADY activity.
- It will usually request that the activity also return a file that is considered "ready" for further processing, if there is one.
- Both these operations can be done in one call to WAIT\_FILESREADY when

you use the default \*PUTGET value for the OPERATION parameter.

It is vital to understand that in that one execution of the WAIT\_FILESREADY activity like this, the file path added to the cache and the file path retrieved from the cache (if any) will NEVER refer to the same file. Furthermore, there will ALWAYS be a delay between adding a file to the cache and a later execution of WAIT\_FILESREADY reporting that it has reached "ready" state – indeed this delay is a part of the very intention of this activity.

If you have understood this, then you will most likely also appreciate that at any point in time there may well be one or more files in the cache that have not yet been assessed as "ready" and that await further processing. Understanding this is very important, because when your processing sequence completes the loop that contains the WAIT\_FILESREADY activity, it has not necessarily processed all the files that were added to the cache.

Once you have understood these two things, then you are a well on the path to being able to implement the WAIT\_FILESREADY activity successfully in your solution.

You can refer to the **EXAMPLE\_WATCH01** Processing Sequence for an example of using the WAIT\_FILESREADY activity.

Note: This activity requires that the LANSA Integrator JSM is executing on a JVM at Java 7 or above.

Note: the file status and metrics checked by the WAIT\_FILESREADY activity to determine "ready" state will be sufficient for many solutions. However some applications may call for checks that are not performed by WAIT\_FILESREADY. You should make sure you understand the characteristics of the environment in which you propose to use WAIT\_FILESREADY to make sure that the activity is suitable for your purpose.

## **Restarting the WAIT\_FILESREADY activity**

The WAIT\_FILESREADY activity supports processing sequence restart if an error occurs while it is active. However, the contents of the files cache are not automatically restored. Consequently, it is possible that some previously cached files may have been lost when the activity restarts. Your solution may need to take account of this. Often, when used with the WATCH\_DIRECTORY iterator activity with \*YES specified for the EXISTINGCONTENTS parameter, for example, the loss of the cache will not be significant, and no special action will

be required.

## **INPUT Parameters:**

**OPERATION**: Optional

This parameter specifies the action(s) that the activity is to perform. The default value used is \*PUTGET.

You may specify one of the following:

\*PUT: adds the file specified by the FILEPATHPUT parameter to the cache, using the values specified by the INTERVALS, READYREPEATS and EVALUATE parameters as the values that will apply to determing the "ready" status of this file.

\*GET: retrieves the next available file from the files cache that has reached "ready" status (if there is one), removes it from the cache and puts its file path into the FILEPATHGET output parameter.

\*PUTGET: combines the functions of both the \*PUT and \*GET operation

\*GETNOTREADY: retrieves a file from the files cache without evaluating its "ready" status (if there is one), removes it from the cache and puts its file path into the FILEPATHGET output parameter. When this operation is used, the files are retrieved in FIFO order. You might use this operation in a loop (while COUNTWAITING is not zero, for example) when a processing sequence is ending to take some action of your choosing for any files that remain in the files "ready" cache.

\*LOG: writes a processing sequence log entry for each file remaining in the cache. The log entry identifies the file and a value that indicates the last reason that the file was evaluated to be not "ready". This operation is intended for use during the design phase of your solution – it should not normally remain in a completed solution.

## FILEPATHPUT: Optional

If \*PUT or \*PUTGET is specified for the OPERATION parameter, this parameter must specify the full path to the file that is to be added to the files cache.

## **INTERVALS:** Optional

If \*PUT or \*PUTGET is specified for the OPERATION parameter, this parameter may specify the intervals (in seconds) at which the file's "ready" status will be evaluated. In other words, the file's status will not be evaluated until at least this number of seconds has elapsed since the file was added or since the last status check was performed. If you do not specify a value, the default value used is 40 (seconds). **READYREPEATS**: Optional

If \*PUT or \*PUTGET is specified for the OPERATION parameter, this parameter may specify the number of consecutive times the file must be assessed as "ready" before it is finally released from the cache in response to a \*GET or \*PUTGET operation.

If you do not specify a value, the default value used is 3 (three). Note that, when used with the default value of 40 for the INTERVALS parameter, this will result in at least two minutes elapsed before the activity will consider the file as having reached "ready" status, even if no changes to the file occurred after it was added to the files cache.

## **EVALUATE:** Optional

If \*PUT or \*PUTGET is specified for the OPERATION parameter, this parameter specifies the criteria or file attributes that are considered when subsequently evaluating the file's "ready" status. You may specify one or more of the following values (each value should be separated by at least one space):

\*ZEROLEN: this test stipulates that a zero-length file will NOT be considered to be "ready".

\*SIZE: this test requires that the file's size must not have changed between consecutive status checks in order for the file to be eligible for "ready" state.

\*LASTCHANGED: this test requires that the file's last-modified date and time must not have changed between consecutive status checks in order for the file to be eligible for "ready" state.

\*READABLE: this test checks that a file exists and that the job has appropriate privileges that would allow it to open the file for reading.

\*LOCKABLE: this test attempts to acquire an exclusive lock on the file (the lock is immediately released, if obtained). If the lock cannot be acquired the file will NOT be considered to be "ready".

The default value includes all the criteria listed above. All the specified criteria must be satisfied for the file to be considered "ready" and therefore eligible to be released from the cache on a \*GET or \*PUTGET operation.

NB: You must specify quote marks around the list of value(s) to distinguish them from built-in variable names.

OUTPUT Parameters: RESULTPUT: If \*PUT or \*PUTGET is specified for the OPERATION parameter, this output parameter will contain the result of the \*PUT operation. It will contain one of the following values:

OK: the file was successfully added to the files cache.

ER: an error occurred while attempting to add the file to the files cache.

Note that the error condition does not result in the activity ending with an error status – in other words, a CATCH processing sequence directive will not be fired for this condition. You need to test the value of the RESULTPUT operation if you wish to specifically handle the 'ER' case.

## **RESULTGET:**

If \*GET, \*PUTGET or \*GETNOTREADY is specified for the OPERATION parameter, this output parameter will contain the result of the \*GET or \*GETNOTREADY operation. It will contain one of the following values:

OK: a file in the cache was evaluated as "ready" (\*GET and \*PUTGET operations only), removed from the cache and its file path was put into the FILEPATHGET output parameter.

EMPTY: no file was retrieved from the cache because the files cache is presently empty.

NOTREADY: no file was retrieved from the cache because no files in the cache were evaluated as "ready" (\*GET and \*PUTGET operations only).

Your processing sequence should, at the minimum, test for the 'OK' result code and treat it as the trigger for further processing of the file that was retrieved.

#### FILEPATHGET:

If \*GET, \*PUTGET or \*GETNOTREADY is specified for the OPERATION parameter and if the RESULTGET output parameter contains 'OK', then this output parameter will contain the file path of the file that was retrieved from the cache.

## **REASONNOTREADY**:

If \*GETNOTREADY is specified for the OPERATION parameter and if the RESULTGET output parameter contains 'OK', then this output parameter will contain a value that indicates the reason that the file was last evaluated to be not "ready". The reason codes can include \*NOTEVALUATED. \*ZEROLEN, \*SIZE, \*LASTCHANGED, \*READABLE, \*LOCKABLE and \*READYREPEATS.

## **COUNTWAITING**:
Irrespective of the requested operation or its result, this output parameter will contain a count of the number of files remaing in the files cache.

# WATCH\_DIRECTORY

This is an iterator activity that watches a specified directory for new, changed and/or deleted files and sub-directories. Upon each iteration, it will output details of the file or sub-directory affected, if any.

The processing logic nested beneath the WATCH\_DIRECTORY activity is repeated each time that a file or sub-directory is added, changed or deleted or each time the specified wait time (WAIT parameter) is exceeded.

A stereotypical application of this activity is where you have a particular directory in which other users or applications can place new files that need to be processed by your solution – a"dropbox" directory. This activity enables you to implement such a solution by watching that directory and receiving notifications when a file is added.

There remains an important consideration for any application that performs this type of processing, and especially when doing so without complete understanding and/or control over the other users or applications that are submitting files for processing.

Your solution will receive a notification of a \*CREATE event, for example, as soon as a file is added to the watched directory. But that does not necessarily mean the file is ready or available for your solution to process. For example, the source application may still be writing it, or even may have created it with a temporary file name before renaming it when the file is complete. Your solution may need a means to decide when a new (or changed) file is ready for it to process.

LANSA Composer provides the WAIT\_FILESREADY activity to assist with many such situations. If you are using the WATCH\_DIRECTORY activity, you may need to use it in conjunction with the WAIT\_FILESREADY activity for a robust solution.

You can refer to the **EXAMPLE\_WATCH01** Processing Sequence for an example of using the WATCH\_DIRECTORY activity in conjunction with the WAIT\_FILESREADY activity.

Note: This activity requires that the LANSA Integrator JSM is executing on a JVM at Java 7 or above.

Note: Unlike many other iterator activities, this activity will iterate indefinitely unless you take specific action in the iterator block to exit

the iteration. It is designed to be used in a potentially long running processing sequence that perpetually waits on a message arriving at the specified message queue and then processes the message or initiates another processing sequence to process the message. For more information refer to Considerations for Extended Duration Processing Sequences.

# **Restarting the WATCH\_DIRECTORY activity**

The WATCH\_DIRECTORY activity supports processing sequence restart if an error occurs while it is active. However, it is possible that some previously queued file system events may have been lost when the activity restarts. Your solution may need to take account of this.

If your solution uses the "drop-box" approach where the directory should always be empty of anything but unprocessed files, then you will probably specify \*YES for the EXISTINGCONTENTS parameter. In this case the existing contents of the specified directory will be enumerated and iterated when the processing sequence restarts, and you will usually need to take no further action with respect to the processing sequence restart support.

# **INPUT Parameters:**

# **DIRECTORY**: Required

This parameter must specify the full path to the directory to be watched.

# **RECURSIVE**: Optional

This parameter specifies whether descendant directories of the specified directory are also watched. The default value is \*NO which means that descendant directories are not watched. You may specify \*YES if you want the activity to watch the descendant directories. If you specify \*YES and you are watching for \*CREATE events (see the WATCHEVENTS parameter), then any new sub-directories that are created while this iterator activity is active will also be watched.

# EXISTINGCONTENTS: Optional

This parameter specifies whether the activity enumerates and iterates for the existing contents of the specified directory when the activity starts up or when it is restarted.

The default value is \*YES which means that the existing directory contents will be processed and an \*EXISTS event generated for each existing file or sub-directory. If your solution uses the "drop-box" approach where the directory should always be empty of anything but unprocessed files, then this

is the behaviour you probably will want.

Alternatively, specify \*NO if you do not want the activity to process the existing directory contents. In this case, your solution will receive notifications only for files that are added, changed or deleted after the iterator activity begins processing.

# WAIT: Optional

Specifies the length of time (in seconds) that the activity will wait, on each iteration, for a watched event to occur in the specified directory.

The default value is 20 (twenty) seconds. This default value has been chosen to be less than the default controlled end delay times used on IBM i system commands such as ENDJOB (End Job). You should consider also the corresponding delay times used with IBM i system commands such as ENDSBS (End Subsystem) or PWRDWNSYS (Power Down System). This enables your processing sequence to have the opportunity to check the value of the \*SHUTDOWN built-in variable and end in a controlled fashion if one of those commands is issued.

You may specify the special value of 0 (zero) which means that the activity does not wait for a watched event to occur in the specified directory.. Instead the next iteration is performed immediately. Use of this value is not generally recommended with the WATCH\_DIRECTORY activity as it may result in your long-running processing sequence being heavily CPU intensive, instead of simply entering an idle/wait state when it has no work to perform.

## WATCHEVENTS: Optional

This parameter specifies the types of file system events that the activity watches for in the specified directory. The default value is \*CREATE which means that the activity watches only for new files or sub-directories.

Alternatively you may specify one or more of the following values (each value should be separated by at least one space):

\*CREATE

\*MODIFY

\*DELETE

You must specify quote marks around the value(s) to distinguish them from built-in variable names.

Note that if you specify \*YES for the EXISTINGCONTENTS parameter, then your solution may also receive \*EXISTS events, irrespective of your

choices for the WATCHEVENTS parameter.

#### **OUTPUT Parameters:** EVENTOUT:

Upon each iteration, the value of this output parameter will indicate whether a watched file system event occurred, and, if so, the type of event. It may have one of the following values:

\*NONE: indicates that no watched event occurred. This value is returned whenever the wait time specified on the WAIT parameter elapsed with no watched file system events for the specified directory.

\*EXISTS: indicates that the remaining output parameter values refer to a file or sub-directory that exists in the specified directory when the activity starts up or when it is restarted. This value will occur only if you specify \*YES for the EXISTINGCONTENTS parameter.

\*CREATE: indicates that the remaining output parameter values refer to a new file or sub-directory that has been created in the specified directory.

\*MODIFY: indicates that the remaining output parameter values refer to a file or sub-directory that has changed in the specified directory.

\*DELETE: indicates that the remaining output parameter values refer to a file or sub-directory that has been deleted in the specified directory. Only the value of the FILEPATHOUT parameter is valid for this event.

\*OVERFLOW: indicates that the file system may have reported events faster than they can be retrieved or processed by this activity. In this case, some events may have been lost. If you process this event, you might use it to re-examine the contents of the directory being watched. The remaining output parameters values are not filled when this event occurs. Your solution may receive this event irrespective of the value(s) specified for the WATCHEVENTS parameter.

## FILEPATHOUT:

Upon each iteration, for events \*EXISTS, \*CREATE, \*MODIFY and \*DELETE, this parameter will contain the full path of the file or subdirectory affected.

## **ISDIROUT**:

Upon each iteration, for events \*EXISTS, \*CREATE and \*MODIFY, this parameter will contain 'Y' if the event refers to a sub-directory. Otherwise the event refers to a file.

## SIZEOUT:

Upon each iteration, for events \*EXISTS, \*CREATE and \*MODIFY, this parameter will contain the size in bytes of the affected file or sub-directory. **LASTMODOUT**:

Upon each iteration, for events \*EXISTS, \*CREATE and \*MODIFY, this parameter will contain the last modified date and time for the affected file or sub-directory.

# WATCH\_DTAQ

This is an iterator activity. It will repeatedly attempt to receive data from a named data queue and on each iteration output the data received, if any.

The processing logic nested beneath the WATCH\_DTAQ activity is repeated each time that data is received or each time the specified wait time (DTAWAIT parameter) is exceeded (if no data is available to receive).

This activity treats data received from a data queue as character data. Using this activity to interact with applications that send or receive binary data to or from the data queue is not supported.

Upon each iteration in which the activity receives an entry from the named data queue, the data content (and possibly sender information) is available through the output parameters of the activity.

When no data is available to be received (after waiting for the interval specified by the DTAWAIT parameter, if used), the activity iterates normally but the output data parameters are not filled. The value of the RCVCOUNT output parameter may be used to determine whether an entry was received. This behavior allows your solution to perform processing periodically whether data is received or not. For example, your processing sequence might check the value of the \*SHUTDOWN built-in variable in order to permit the long running process to be ended in a controlled fashion.

Refer also to the companion DTAQ\_CLEAR, DTAQ\_RECEIVE and DTAQ\_SEND activities that allows you to send and receive data to and from and to perform other operations on a named data queue.

Note: Data queues are persistent objects but their data content can be lost or corrupted during a system failure. Backup, recovery and maintenance of data queues used in your LANSA Composer solution is your responsibility – you should make provision for this in your implementation plans.

Note: Unlike many other iterator activities, this activity will iterate indefinitely unless you take specific action in the iterator block to exit the iteration. It is designed to be used in a potentially long running processing sequence that perpetually waits on a message arriving at the specified message queue and then processes the message or initiates another processing sequence to process the message. For more information refer to Considerations for Extended Duration Processing Sequences .

# **IBM i Platform Notes**

Data queues are a native operating system feature on the IBM i server. The full range of capabilities of this activity is only available on IBM i servers.

(The implementation of this activity on an IBM i server uses the QRCVDTAQ system API.)

Use of keyed data queues with this activity is not presently supported.

Refer to the description of the DTAQ\_SEND activity for more complete IBM i Server Platform Notes relating to the suite of DTAQ activities. Refer to IBM i documentation concerning data queues for further information about the operating system features upon which this activity depends.

# Windows Server Platform Notes

Although data queues are a native feature of the IBM i server, this activity and the companion DTAQ\_CLEAR, DTAQ\_RECEIVE and DTAQ\_SEND activities are available on Windows servers too, but with a reduced feature set and with other restrictions and caveats.

(The implementation of this activity on a Windows server uses the emulated data queue support in the LANSA run-time, using the RCV\_FROM\_DATA\_QUEUE built-in function.)

The following notes and restrictions apply to using this activity on a Windows server:

- The DTAQLIB parameter is not used and its value will be ignored. (You may use the DTAQPATH parameter to specify a location for the data queue, if required.)
- Data queues are stored in pairs of files in the Windows server file system with .EDQ and .LDQ file extensions. The .EDQ file contains the data queue definition and entries. The .LDQ files are used to logically lock the data queue during receive operations. The file names are determined by the LANSA run-time support used by this activity. The files can be manually cleared or deleted by deleting the associated files. Alternatively the DTAQ\_CLEAR activity may be used for this purpose.
- The location of the .EDQ and .LDQ data queue storage files is determined by the value of the DTAQPATH parameter. Refer to that parameter description for further information.

• Refer to the description of the DTAQ\_SEND activity for more complete Windows Server Platform Notes relating to the suite of DTAQ activities.

NOTE: On Windows servers, the specific implementation used means that the activity is unable to handle certain types of exceptions that may be considered unexceptional in other circumstances. In such cases the processing sequence containing the activity may simply end abnormally without logging diagnostic information in the Processing Sequence log. This behavior differs from most other LANSA Composer activities and from the IBM i implementation of this activity.

You should be aware of this possibility and be prepared to diagnose such issues in other ways. For example, the x\_err.log generated by the LANSA run-time may assist with diagnosing such conditions.

#### **INPUT Parameters:**

#### **DTAQ**: Required

This parameter must specify the name of the data queue from which the data is to be received.

#### DTAQLIB: Optional

When running on IBM i servers, this parameter may specify the name of the library containing the data queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named data queue.

When running on a Windows server, this parameter is not used and its value is ignored. You may use the DTAQPATH parameter to specify the location for the data queue, if required.

#### **DTAQPATH**: Optional

When running on Windows servers, this parameter may specify the path to the directory in which the data queue .EDQ and .LDQ files are stored. If not specified, the default location is determined by the LANSA run-time according to the effective value of the DPTH= X\_RUN parameter.

When running on an IBM i server, this parameter is not used and its value is ignored. You may use the DTAQLIB parameter to specify the location for the data queue, if required.

#### **DTAWAIT**: Optional

Specifies the length of time (in seconds) that the activity will wait, on each

iteration, for data to arrive on the data queue.

The default value is 20 (twenty) seconds. This default value has been chosen to be less than the default controlled end delay times used on IBM i system commands such as ENDJOB (End Job). You should consider also the corresponding delay times used with IBM i system commands such as ENDSBS (End Subsystem) or PWRDWNSYS (Power Down System). This enables your processing sequence to have the opportunity to check the value of the \*SHUTDOWN built-in variable and end in a controlled fashion if one of those commands is issued.

You may specify the special value of 0 (zero) which means that the activity does not wait for the arrival of an entry if one is not immediately available when the activity is processed. Instead the next iteration is performed immediately. Use of this value is not generally recommended with the WATCH\_DTAQ activity as it may result in your long-running processing sequence being heavily CPU intensive, instead of simply entering an idle/wait state when it has no work to perform.

You may also specify the special value \*MAX which means that the activity waits indefinitely for data to arrive. Use of this value is not generally recommended with the WATCH\_DTAQ activity as it will prevent any possibility of effecting controlled end of the processing sequence.

#### **DTARMV**: Optional

Specifies whether the data received by the activity is removed from the data queue. The default is \*YES, which means the data is removed. You may specify \*NO to cause the data not to be removed from the data queue. If you do so, the data can be received again by this activity in the same or another Processing Sequence. You should take care using \*NO with the WATCH\_DTAQ activity – unless you design your solution to prevent it, your processing sequence will loop, repeatedly processing the same data queue entry.

#### **DTAQELEN:** Optional

On an IBM i server, this parameter is not required and its value will be ignored.

On a Windows server, this parameter is required and it must specify the length of the data queue entries. The activity supports data lengths between 1 and 1024 inclusive.

On a Windows server, the first send (DTAQ\_SEND) operation for a data queue will automatically create the data queue and will set its entry length.

The length specified for subsequent operations to the same data queue must match the length specified on the initial operation that created the data queue. (On Windows servers, LANSA Composer will actually use the next greatest multiple of 256 – for example, if you specify 700 for this parameter, LANSA Composer will actually use the value 768.)

# **OUTPUT Parameters:**

## **RCVCOUNT**:

The received entry count. Upon each iteration, if data is received successfully, the value will be one (1). Otherwise the value will be zero (0).

#### **RCVDATALEN:**

Upon each iteration, if data is received, this parameter contains the length of the data received.

(On a Windows server, the activity cannot determine the actual data length received, and the value returned in this parameter is the same as the value provided in the DATAQELEN input parameter.)

#### RCVDATARCVDATA02 RCVDATA03 RCVDATA04:

Upon each iteration, if data is received, these parameters contains the data. Each parameter receives up to 256 bytes of data. The first parameter, RCVDATA, contains the first up to 256 bytes of data. The remaining parameters are filled only if you are using data queue entries of length greater than 256, 512 and 768 respectively.

#### SENDJOB SENDUSER SENDJOBNBR:

Upon each iteration, if data is received, these output parameters may contain the job name, user name and the job number of the job that sent the data to the data queue.

On an IBM i server, these values will be available only if

SENDERID(\*YES) was specified when the data queue was created using the CRTDTAQ (Create Data Queue) command.

On a Windows server, the sending job information is not available and these parameters will not be filled.

# WATCH\_MSGQ

This is an iterator activity. It will repeatedly attempt to receive a message from a named message queue on the IBM i server and on each iteration output the details of the message received, if any. It is supported only on IBM i servers.

The processing logic nested beneath the WATCH\_MSGQ activity is repeated each time that a message is received or each time the specified wait time (MSGWAIT parameter) is exceeded (if no message is available to receive).

Upon each iteration in which the activity receives a message from the named message queue, a range of attributes of the message are available through the output parameters of the activity, including message text, message data, message identifier and sender information.

When no message is available to be received (after waiting for the interval specified by the MSGWAIT parameter, if used), the activity iterates normally but the output parameters containing message attributes are not filled. The value of the RCVCOUNT output parameter may be used to determine whether a message was received. This behavior allows your solution to perform processing periodically whether a message is received or not. For example, your processing sequence might check the value of the \*SHUTDOWN built-in variable in order to permit the long running process to be ended in a controlled fashion.

Refer also to the companion MSGQ\_RECEIVE and MSGQ\_SEND activities that allows you to send and receive single messages to or from a named message queue. Refer to IBM i documentation concerning message queues, message descriptions and message files for further information about the operating system features upon which this activity depends.

Note: Unlike many other iterator activities, this activity will iterate indefinitely unless you take specific action in the iterator block to exit the iteration. It is designed to be used in a potentially long running processing sequence that perpetually waits on a message arriving at the specified message queue and then processes the message or initiates another processing sequence to process the message. For more information refer to Considerations for Extended Duration Processing Sequences .

INPUT Parameters: MSGQ : Required This parameter must specify the name of the message queue from which the messages are to be received.

#### MSGQLIB: Optional

This parameter may specify the name of the library containing the message queue. If not specified, a default of \*LIBL is used which means the library list is used to locate the named message queue.

#### **MSGWAIT**: Optional

Specifies the length of time (in seconds) that the activity will wait, on each iteration, for a message to arrive in the message queue.

The default value is 20 (twenty) seconds. This default value has been chosen to be less than the default controlled end delay times used on IBM i system commands such as ENDJOB (End Job). You should consider also the corresponding delay times used with IBM i system commands such as ENDSBS (End Subsystem) or PWRDWNSYS (Power Down System). This enables your processing sequence to have the opportunity to check the value of the \*SHUTDOWN built-in variable and end in a controlled fashion if one of those commands is issued.

You may specify the special value of 0 (zero) which means that the activity does not wait for the arrival of a message if one is not immediately available when the activity is processed. Instead the next iteration is performed immediately. Use of this value is not generally recommended with the WATCH\_MSGQ activity as it may result in your long-running processing sequence being heavily CPU intensive, instead of simply entering an idle/wait state when it has no work to perform.

You may also specify the special value \*MAX which means that the activity waits indefinitely for a message to arrive. Use of this value is not generally recommended with the WATCH\_MSGQ activity as it will prevent any possibility of effecting controlled end of the processing sequence.

#### **MSGRMV**: Optional

Specifies whether each message received by the activity is removed from the message queue. The default is \*YES, which means the message is removed. You may specify \*NO to cause the message not to be removed from the message queue. If you do so, the system considers the message to be "old" and it will not be received again by this activity. If your processing sequence leaves "old" messages on the message queue, it is your responsibility to remove them by other means such as by executing the CLRMSGQ operating system command.

#### OUTPUT Parameters: RCVCOUNT:

The received message count. Upon each iteration, if data is received successfully, the value will be one (1). Otherwise the value will be zero (0).

# **RCVMSGSEV**:

If a message is received, this parameter contains the severity code of the received message. Impromptu messages sent by the MSGQ\_SEND activity will usually have a severity code of '00'. For pre-defined messages, the severity code is defined by the message description.

#### **RCVMSGTYPE**:

If a message is received, this parameter contains the message type of the received message. Messages sent by the MSGQ\_SEND activity will have a message type of \*INFO. For messages sent by other means, the message type is specified by the sending program.

#### **RCVMSGID**:

If a message is received, this parameter contains the message identifier of the received message. An impromptu message has no message identifier. Otherwise, this value identifies the message description in the message file identified by the RCVMSGF and RCVMSGFLIB parameters.

#### **RCVMSGF**:

If a message is received, this parameter contains the name of the message file of the received message. An impromptu message has no message file name. Otherwise, this value identifies the message file name containing the message description used to send the message.

#### **RCVMSGFLIB**:

If a message is received, this parameter contains the name of the library containing the message file, if any.

#### **RCVMSGTXT**:

If a message is received, this parameter contains the first level message text of the received message. For a pre-defined message, this includes the message data fields that were substituted for substitution variables in the text before the message was sent. A maximum of 256 characters of the first level text will be returned.

## **RCVSECLVL**:

If a message is received, this parameter contains the second level message

text of the received message, if any. An impromptu message has no second level message text. For a pre-defined message, this includes the message data fields that were substituted for substitution variables in the second level text before the message was sent. A maximum of 256 characters of the second level text will be returned.

# **RCVMSGDTA**:

If a message is received, this parameter contains the message data field values of the received msg. An impromptu message has no message data. Otherwise this contains a single string of the message data values used to send the message. A maximum of 256 characters of message data will be returned.

#### **SENDJOB**:

If a message is received, this parameter contains the name of the job that sent the message.

## SENDUSER:

If a message is received, this parameter contains the user name of the job that sent the message.

#### **SENDJOBNBR**:

If a message is received, this parameter contains the job number of the job that sent the message.

#### **SENDDATE**:

If a message is received, this parameter contains the date that the message was sent. The date is in ISO format - that is, CCYY-MM-DD.

#### SENDTIME:

If a message is received, this parameter contains the time that the message was sent. The time is in ISO format - that is, HH:MM:SS.

# XML\_SPLIT

This activity can split an XML document into multiple documents at a specified element name, with certain constraints and caveats as further described below.

For most routine XML document processing, this approach is NOT necessary or recommended. This activity is provided primarily for use in cases involving XML documents that are so exceptionally large that they cannot be efficiently processed in one step due to memory constraints. An exceptionally large XML file will still take a long time to process with this activity - you should verify the performance with your files and in your operating environment before designing a solution that relies on this activity.

In order to optimise the activity, it processes the source XML document without reference to any associated schema or DTD. The split documents created by this activity should be *well-formed* (\*) XML (providing the source XML document is well-formed). However, it cannot be guaranteed that the split documents are *valid* (\*) with reference to the source document schema or DTD. In addition, depending on the structure of the source +document and upon the split element specified, the split documents may not always contain all the available data in the source document.

(\*) This description assumes you understand the distinction between the terms "well-formed" and "valid" in relation to XML documents. If in doubt, please refer to appropriate XML references.

In addition, please understand that XML is capable of representing an infinite variety of document structures. This activity may not be capable of splitting every source document in the way that you intended or expected. In order to keep the activity simple to use while enabling it to be useful for the largest range of likely scenarios, the activity makes certain assumptions about how to split documents and what antecedent data to retain or duplicate in the split documents. You should thoroughly test it with representative samples of the XML documents with which you intend to use it before designing a solution that relies on this activity.

In addition, you must be aware of the following further limitations and caveats:

- The DOCTYPE of the source XML document (for XML documents that reference a DTD) is NOT output to the split documents.
- Certain other parts of an XML document may similarly NOT be output to the split documents, in particular CDATA sections.

- White space (including line feeds and carriage-returns) in the source XML document will be lost from the split documents. In most cases, this simply makes the XML less-readable when viewed with a non-XML aware text editor or viewer.
- Finally, it should be evident that the XML\_SPLIT activity cannot and does not adjust any XML element or attribute values that represents an aggregation of other data in the source XML document. For example, if an <ORDERS> element includes values that represent an aggregation of the order lines contained within it (for example, a count of the number of order lines or a total of the order quantities or values), then that data would be repeated without modification in each split document that arises from that <ORDERS> element.

The output split document paths and names are generated according to the values specified or assumed for the XMLSPLITPATH and XMLSPLITROOT parameters. If documents of the generated name(s) exist in the target location, they will be replaced by this activity.

#### **INPUT Parameters:**

#### **XMLFILE** : Required

This parameter specifies the path and name of the XML document file to be split.

#### XMLSPLITELEMENT : Optional

This parameter may specify the name of the XML element at which the source XML document is to be split. If not specified, a default of \*FIRST is assumed. \*FIRST means the name of the first XML element INSIDE the root element (ie: NOT the root element) is used as the split element name.

Note that, for convenience, the element name comparisons are performed without regard to case. (In strict XML terms, this is potentially ambiguous as XML element names are case-sensitive.) Also the undecorated element name is used for comparisons - that is, excluding any namespace qualifiers.

Each instance of an element (other than the root element) that matches the specified name (and that is not contained within another instance of an element with the same name) may trigger a new split XML document, according to the value specified for the XMLSPLITELEMENTSMAX parameter. Each split document will contain an XML structure that includes:

• all the antecedent elements of the split element and all preceding elements that they contain

- PRECEDING sibling elements, ie: elements (other than preceding instances of the split element) with the same parent element
- the instance(s) of the split element and all elements that they contain.

# XMLSPLITELEMENTSMAX : Optional

This parameter may specify the maximum number of split element instances per split document. If not specified, a default of 1 is assumed. The activity will place "sibling" instances of the split element into the same split document, up to the number specified by this parameter. When the parent element is closed, the split document is closed whether or not the maximum is reached. Further instances of the split element will trigger another split document for which the count restarts at one.

# XMLSPLITPATH : Optional

This parameter may specify the path in which the split XML document files are to be created. If not specified, a default of \*SAME is assumed. \*SAME means the split XML document files will be created in the same location as the input XML document file.

## XMLSPLITROOT : Optional

This parameter may specify the root file name and the file extension for the split XML document files. If not specified, a default of \*SAME is assumed. \*SAME means the activity will use the file name and extension of the input XML document file as the root file name and the file extension for the split XML document files. The activity will append a sequential number to the root file name to make each split XML document file name. For example, if you specify 'ORDER.xml'as the value for this parameter and the input file is split into three XML document files, then they would have the names 'ORDER1.xml', 'ORDER2.xml' and 'ORDER3.xml'.

## XMLSPLITNOTRACE : Optional

This activity uses LANSA Integrator's XMLReaderService and XMLWriterService. These services may generate particularly large LANSA Integrator trace files, especially as this activity is typically used to process exceptionally large XML files. More so than many activities, the performance of this activity will be dramatically improved if LANSA Integrator tracing is not in effect while it is executing. The default value for this parameter, if not specified, turns the LANSA Integrator tracing OFF. If you want to use LANSA Integrator tracing, then specify 'NO' for this parameter. You are advised to do this only during design and testing and only with relatively small sample input XML files. Note that this does NOT necessarily enable tracing - it simply makes it subject once more to LANSA Composer's *System Settings*.

#### OUTPUT Parameters: XMLSPLITCOUNT:

Upon successful completion this parameter will contain the count of split documents created.

## XMLSPLITLIST:

Upon successful completion this parameter will contain a list of the full file paths of the split documents created.

# XML\_QUERY

The XML\_QUERY activity permits a LANSA Composer solution to selectively interrogate values contained in an XML document using XML Path Language (XPath) expressions.

XPath is an industry-standard syntax for constructing path expressions to select nodes in an XML document. It is beyond the scope of this document to provide a detailed description of XPath syntax. However, you can refer to the following heading for an introduction and examples:

• Quick Guide to XPath expressions for use with the XML\_QUERY activity

The XML\_QUERY activity is not suitable or intended for and usually would not be used to process the entire contents of an XML document. Rather it is intended for limited and selective interrogation of particular values from the XML document. In some applications, for example, it may be necessary for your LANSA Composer solution to determine certain key values from the XML document in order to decide how to proceed with further processing or for use in the course of further processing.

As an example, suppose you are building a solution that receives and processes sales orders in an agreed XML format.

One part of your solution might process the sales order XML document in its entirety, probably using a LANSA Composer Transformation Map to read the contents and update your application database.

However, in a separate step, your solution is required to email an acknowledgement of the order. In order to do so, the most convenient means to access the customer's return email address is directly from the corresponding element value in the original sales order XML document.

In such circumstances, your solution could use the XML\_QUERY activity to selectively and efficiently address and retrieve just the value of the element (or attribute) in the sales order XML document that contains the customer's return email address and then pass that to the MAIL\_SEND activity or whatever means your solution uses to generate and send the acknowledgement.

NOTE: The XML\_QUERY activity loads the entire XML document into memory when processing your queries. Application performance can degrade when used with exceptionally large XML files.

The supplied processing sequence EXAMPLE\_XML01 implements the

XML\_QUERY activity to interrogate the supplied tutorial orders XML file, TUTorder.xml. This is a working example that you may wish to use as a starting point for your exploration of the XML\_QUERY activity.

# **INPUT Parameters:**

# XMLFILE : Required

This parameter specifies the path and name of the XML document file whose contents are to be queried.

Note that the XML\_QUERY activity is optimised to avoid reloading the XML document if it is the same as the preceding usage in the same Processing Sequence.

# **XMLOPTIONS** : Optional (Advanced)

This parameter allows you to specify options that affect the way the XML document is loaded and queried.

You may specify one or more of the values described below. Each value should be separated by at least one space, and you must specify quote marks around the complete string of value(s) to distinguish them from built-in variable names.

# 1. \*VALIDATE

By default the XML document is parsed WITHOUT validating it against any DTD or schema it may reference. In this mode, the XML document is required to be well-formed, but conformance to the DTD or schema is NOT checked. For many typical applications of the XML\_QUERY activity, validation is an unnecessary overhead. However, you can specify this option if you require validation to be performed.

## 2. \*NAMESPACEAWARE

By default the XML document is parsed in a non-namespace-aware mode. For most cases, this simplifies the form of the XPath expressions necessary to perform a given query.

In some more complex documents (and especially for documents in which more than one namespace is referenced) it may be necessary to specify this option in order to load the document in namespace-aware mode.

Note that specifying this option will affect the form of XPath expressions necessary to successfully perform a given query. For more information, refer to:

• XML Namespaces and How They Affect XPath Expressions for the XML\_QUERY Activity

NOTE: If the XML document specified by the XMLFILE parameter is already loaded by the preceding use of the XML\_QUERY activity in the same processing sequence, the options specified in the XMLOPTIONS parameter will not be effective even if they changed from the earlier use.

# CURRENTNODE : Optional (Advanced)

This parameter can be used to set the "current" node in the presently loaded XML document. The "current" node is the node to which the XPath expression in the QUERYNODES parameter will apply.

By default, the current node is the document node. The special value (and default) \*DOCUMENT specifies the document node. Providing the XPath expression used in the QUERYNODES parameter is formulated appropriately, it can be executed against the document node, and in many cases this will be all that is needed.

However, in more complex applications of the XML\_QUERY activity, particularly when used iteratively, you may specify an XPath expression that identifies a single node that is to be the current (context) node for the QUERYNODES expression. In particular, the XPath expressions generated when the special \*XPATH or \*XPATH\_CONCISE values are used (in the QUERYNODESVALUE1 through QUERYNODESVALUE5 parameters) may be used in subsequent iterations as the current node for further queries.

If you use this parameter, you should specify an appropriately formulated XPath expression that identifies a single node in the document that is to be the "current" node. For more information about XPath expressions used with the XML\_QUERY activity, refer to:

• Quick Guide to XPath expressions for use with the XML\_QUERY activity

# **QUERYNODES** : Required

The QUERYNODES parameter must specify an appropriately formulated XPath expression that identifies one or more nodes in the document for which values are to be returned.

The XPath expression is evaluated in the context of the "current" node. By default the "current" node is the document node but it may be altered using the CURRENTNODE parameter.

For more information about XPath expressions used with the XML\_QUERY activity, refer to:

• Quick Guide to XPath expressions for use with the XML\_QUERY activity

Note that if no nodes are selected when the XPath expression is evaluated, it is not treated as an error. Instead the RESULTCOUNT output parameter will be set to zero (0) and the RESULTVALUES1 through RESULTVALUES5 output parameters will not be filled.

## QUERYNODESVALUE1 : Optional

The QUERYNODESVALUE1 parameter identifies the first of up to five values that are to be returned in the RESULTVALUES1 through RESULTVALUES5 output parameters for each "selected" node (that is, each node "selected" by evaluating the XPath expression specified in the QUERYNODES parameter).

This parameter is optional. If the XPath expression specified in the QUERYNODES parameter fully identifies a set of elements or attributes whose value(s) are to be returned, there is no need to specify the QUERYNODESVALUE1 parameter. The default value of \*NODEVALUE specifies that the value of the context node (in this case, each "selected" node) is to be used.

If you specify this keyword, you should specify a further XPath expression that will be evaluated in the context of each "selected" node and that will identify a single element or attribute value relative to that node whose value is to be returned.

For more information about XPath expressions used with the XML\_QUERY activity, refer to:

• Quick Guide to XPath expressions for use with the XML\_QUERY activity

Alternatively, you may specify one of the following special values:

\*LOCALNAME: returns the node name (usually an element or attribute name) of the "selected" node, without any namespace prefix, if present; \*NAMESPACEPREFIX: returns the namespace prefix of the "selected" node;

\*NAMESPACEURI: returns the namespace URI of the "selected" node; \*NODENAME: returns the node name (usually an element or attribute name) of the "selected" node;

\*NODEVALUE: returns the value of the "selected" node, equivalent to using an XPath expression of '.';

\*XPATH:

\*XPATH\_CONCISE: These values both return a generated XPath expression that uniquely identifies the "selected" node within the XML document. The generated XPath expressions mostly use ordinal notation, and so are valid only for the specific node instance in the specific document instance. Such generated expressions can be used, however, to iteratively process a document using further QUERY commands. The second form generates an expression that is more concise (though less human-readable), particularly when the document is being processed in namespace-aware mode. The concise form may be necessary for use with very complex XML documents that may otherwise generate XPath expressions that are longer than can be held in a processing sequence variable.

#### QUERYNODESVALUE2 QUERYNODESVALUE3 QUERYNODESVALUE4 QUERYNODESVALUE5 : Optional

The parameters QUERYNODESVALUE2 through QUERYNODESVALUE5 are optional, but, if specified, they function similarly to the QUERYNODESVALUE1 parameter.

They allow you to specify further XPath expressions that (with the QUERYNODESVALUE1 parameter) identify up to five separate values relative to each "selected" node to be returned in the RESULTVALUES1 through RESULTVALUES5 output parameters for each "selected" node. Unlike the QUERYNODESVALUE1 keyword, these keywords have no default. If you do not specify them, they will not be used and the corresponding output parameter is not filled.

If you do specify these parameters, you must do so contiguously. The activity will stop looking after the first parameter that is not used. For example, if you specify QUERYNODESVALUE1 and QUERYNODESVALUE3, then the latter will be ignored because QUERYNODESVALUE2 was not used.

#### OUTPUT Parameters: RESULTCOUNT:

Upon successful completion this parameter will contain the count of nodes selected by the XPath expression specified in the QUERYNODES parameter.

## **RESULTVALUES1**:

Upon successful completion this parameter will contain a list of the values, for each "selected" node, specified by QUERYNODESVALUE1 parameter.

#### RESULTVALUES2 RESULTVALUES2 RESULTVALUES4 RESULTVALUES5:

Upon successful completion these parameters will contain lists of the values, for each "selected" node, specified by the corresponding QUERYNODESVALUE2 through QUERYNODESVALUE5 parameter, if used.

# **Quick Guide to XPath expressions for use with the XML\_QUERY activity**

There are many XPath resources available on the web. To get you started, you could try the following:

- For one quick and easy introduction to XPath: XPath Tutorial
- Another XPath tutorial can be found here: XPath Tutorial
- XML in a Nutshell A Desktop Quick Reference: Chapter 9 XPath
- For a description of the XPath standard: XML Path Language (XPath) Version 1.0

This document does not intend to provide a definitive description of or reference to XPath expression syntax. However, for those readers who have not used XPath expressions before, following is a brief overview and examples that might help you get started with the XML\_QUERY activity. Refer to the following headings:

- Example XML
- Introduction to XML Path Language (XPath)
- XPath Examples for use with the XML\_QUERY Activity
- XML Namespaces and How They Affect XPath Expressions for the XML\_QUERY Activity

# **Example XML**

The examples provided later in this section will refer to the following simple example XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

<!DOCTYPE Orders SYSTEM "http://www.lansa.com/schemas/tutorder.dtd" >

<Orders>

<SalesOrder SONumber="12345">

<Customer CustNumber="543"> <CustName>ABC Industries</CustName> <Street>123 North St.</Street>

```
<City>Bankstown</City>
<State>NSW</State>
<PostCode>2087</PostCode>
</Customer>
```

```
<OrderDate>2012-11-19</OrderDate>
```

```
<Line LineNumber="1">
<Part PartNumber="123">
<Description>Gasket Paper</Description>
<Price>9.95</Price>
</Part>
<Quantity>10</Quantity>
</Line>
```

```
<Line LineNumber="2">
<Part PartNumber="456">
<Description>Glue</Description>
<Price>13.27</Price>
</Part>
<Quantity>5</Quantity>
</Line>
```

```
</SalesOrder>
```

```
</Orders>
```

# Introduction to XML Path Language (XPath)

XPath is a syntax for constructing path expressions to select nodes in an XML document. To some extent, these path expressions look very similar to path expressions you use when working with the file system on your computer.

In general, XPath recognises seven types of nodes, viz. element, attribute, text, namespace, processing-instruction, comment, and document nodes. In the context of the XMLQueryService we are chiefly concerned with the element and attribute nodes and of course, the document node.

In the example XML document shown above, some of the *elements* are <Orders>, <SalesOrder>, <Customer>, <OrderDate>, <Line> and <Part>, while the *attributes* include SONumber=, CustNumber=, LineNumber= and

PartNumber=.

The following is an example XPath expression that will select the PartNumber= attribute of the first <Part> element in the second <Line> element in the first <SalesOrder> element of the example XML document:

/Orders/SalesOrder[1]/Line[2]/Part[1]/@PartNumber

Note that the selection of the <SalesOrder>, <Line> and <Part> elements in the above example are by ordinal index. In particular, the selection of the <Line> element does NOT refer to the value of the LineNumber= attribute (although that is possible too, as you will see later).

XPath provides a large number of built-in functions that can manipulate and compare values in a variety of ways for more advanced usage. For example, the following expression uses the *contains* built-in function to select all <Part> elements (wherever they occur) whose <Description> element contains the string "Paper":

//Part[contains(Description, "Paper")]

In XPath, you select a node or set of nodes, by following a path or steps. Your XPath expression will often include one or more of the following:

nodename	Selects all nodes with the specified name.
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
•	Selects the current node
••	Selects the parent of the current node

*(@nodename* Selects attributes with the specified name

In XPath, a predicate is a sub-expression contained in square brackets that is used to select a specific node or a node that contains a specific value. The following are some examples of XPath expressions that use predicates:

/Orders/SalesOrder[1]	Selects the first <salesorder> element that is a child of the <orders> element.</orders></salesorder>
/Orders/SalesOrder[last()]	Selects the last <salesorder> element that is a child of the <orders> element. (In the particular instance of the example XML document shown,</orders></salesorder>

```
there is only one <SalesOrder> element and so the result will be the same.)
```

```
//Part[Price<=10.00] Selects <Part> elements, wherever they occur,
whose <Price> element has a value less than or
equal to 10.00.
```

There is much more to know about XPath expressions. If you would like more information, you could start by referring to some of the links provided above.

Important note: XML node names are case sensitive. Your XPath expressions must specify the correct case when specifying element and attribute names. For example, the expression '//salesorder' is NOT the same as '//SalesOrder'. When used with the example XML document shown above, the former expression will FAIL to select ANY nodes, while the latter will select all <SalesOrder> elements, wherever they occur in the document.

# XPath Examples for use with the XML\_QUERY Activity

The following examples use XPath expressions in the parameters of the XML\_QUERY activity to select values from the example XML document shown above.

1. This example will select nothing because XML and XPath are case-sensitive and the wrong case is used to select the <SalesOrder> elements:

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//SALESORDER') QUERYNODESVALUE1('@SONumber')

2.These two examples use alternate implementations to select all <SalesOrder> elements, and return the sales order number for each. Functionally, they are equivalent (when used with the example XML document):

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//SalesOrder/@SONumber')

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//SalesOrder') QUERYNODESVALUE1('@SONumber')

3.Selects all <SalesOrder> elements, and returns the customer number for each:

XML\_QUERYXMLFILE('TUTorder.xml')

QUERYNODES('//SalesOrder') QUERYNODESVALUE1('Customer/@CustNumber')

4.Selects <Customer> elements that have a value of '543' for their customer number and returns the sales order number of the parent <SalesOrder> element:

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//Customer[@CustNumber="543"]') QUERYNODESVALUE1('../@SONumber')

5.Selects all <Part> elements for the <SalesOrder> element(s) with the order number specified and returns the part number and quantity for each:

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//SalesOrder[@SONumber="12345"]/Line/Part') QUERYNODESVALUE1('@PartNumber') QUERYNODESVALUE2('../Quantity')

6.Selects all <Part> elements with a price greater than 2.99 and, for each, returns the order number, the part number, the price, the quantity and calculates and returns the extended value (price \* quantity):

XML\_QUERYXMLFILE('TUTorder.xml') QUERYNODES('//Part[Price>2.99]') QUERYNODESVALUE1('../../@SONumber') QUERYNODESVALUE2('@PartNumber') QUERYNODESVALUE3('Price') QUERYNODESVALUE4('../Quantity') QUERYNODESVALUE5('Price\*../Quantity')

NOTE: the above examples are used in the supplied processing sequence EXAMPLE\_XML01. This is a working example that you may wish to use as a starting point for your exploration of the XML\_QUERY activity.

# XML Namespaces and How They Affect XPath Expressions for the XML\_QUERY Activity

The examples used so far operate on an XML document that contains no explicit namespace declarations and does not make use of namespace prefixes. This is the simplest case, but frequently does not reflect the real world.

Consider this minor alteration to the example XML document that specifies a *default* namespace for the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
```

<!DOCTYPE Orders SYSTEM "http://www.lansa.com/schemas/tutorder.dtd" >

<Orders xmlns="urn:schemas-lansa-com:tutorder.dtd">

```
... etc ...
```

</Orders>

Where a document makes use a single default namespace like this, the easiest approach to formulating XPath expressions for use with it is usually to disregard the namespace. Since only one namespace is used and there are no namespace prefixes present on the node names, you can usually use the same expressions as you would use with the earlier example. Each of the following queries work successfully with the example document that declares the default namespace, providing the document is *not* loaded in namespace-aware mode:

XML\_QUERYXMLFILE('TUTorderNamespaceDefault.xml') QUERYNODES('//SalesOrder/@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespaceDefault.xml') QUERYNODES('//SalesOrder') QUERYNODESVALUE1('Customer/@CustNumber')

XML\_QUERYXMLFILE('TUTorderNamespaceDefault.xml') QUERYNODES('//Customer[@CustNumber="543"]') QUERY NODESVALUE1('../@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespaceDefault.xml') QUERYNODES('//SalesOrder[@SONumber="12345"]/Line/Part') QUERY NODESVALUE1('@PartNumber') QUERY NODESVALUE2('../Quantity')

XML\_QUERYXMLFILE('TUTorderNamespaceDefault.xml') QUERYNODES('//Part[Price>2.99]') QUERY NODESVALUE1('../../@SONumber') QUERY NODESVALUE2('@PartNumber') QUERY NODESVALUE3('Price')

#### QUERY NODESVALUE4('../Quantity') QUERY NODESVALUE5('Price\*../Quantity')

However, in XML documents that use more than one namespace and/or implement namespace prefixes, things can get a little more complicated. Consider the following alternate example XML document and contrast it to the earlier example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

<!DOCTYPE Orders SYSTEM "http://www.lansa.com/schemas/tutorder.dtd" >

<tut:Orders xmlns:tut="urn:schemas-lansa-com:tutorder.dtd">

```
<tut:SalesOrder SONumber="12345">
```

```
<tut:Customer CustNumber="543">
<tut:CustName>ABC Industries</tut:CustName>
<tut:Street>123 North St.</tut:Street>
<tut:City>Bankstown</tut:City>
<tut:State>NSW</tut:State>
<tut:PostCode>2087</tut:PostCode>
</tut:Customer>
```

```
<tut:OrderDate>2012-11-19</tut:OrderDate>
```

```
<tut:Line LineNumber="1">
<tut:Part PartNumber="123">
<tut:Description>Gasket Paper</tut:Description>
<tut:Price>9.95</tut:Price>
</tut:Part>
<tut:Quantity>10</tut:Quantity>
</tut:Line>
```

```
<tut:Line LineNumber="2">
<tut:Part PartNumber="456">
<tut:Description>Glue</tut:Description>
<tut:Price>13.27</tut:Price>
</tut:Part>
```

```
<tut:Quantity>5</tut:Quantity>
</tut:Line>
```

</tut:SalesOrder>

</tut:Orders>

This document contains a namespace declaration and uses the associated namespace prefix on the element names. The use of namespace features and especially of namespace prefixes can complicate the syntax of the XPath expressions necessary for a given query.

Again you should refer to the many resources available on the web concerning XML namespaces and how they affect XPath. One such reference is:

• XML Namespaces and How They Affect XPath and XSLT

The easiest approach to formulating XPath expressions for use with such an instance document is to disregard the namespace(s). If the document is loaded WITHOUT the namespace-aware option (the default mode), then you can use *nearly* the same expressions as you would use with the earlier example. Each of the following queries work successfully with the namespace prefixed version of the document as shown above (note that the namespace prefix is omitted entirely from the XPath expressions):

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') QUERYNODES('/Orders/SalesOrder/@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') QUERYNODES('/Orders/SalesOrder') QUERYNODESVALUE1('Customer/@CustNumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') QUERYNODES('/Orders/SalesOrder/Customer[@CustNumber="543"]') QUERY NODESVALUE1('../@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') QUERYNODES('/Orders/SalesOrder[@SONumber="12345"]/Line/Part') QUERY NODESVALUE1('@PartNumber') QUERY NODESVALUE2('../Quantity')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') QUERYNODES('/Orders/SalesOrder/Line/Part[Price>2.99]') QUERY NODESVALUE1('../../@SONumber') QUERY NODESVALUE2('@PartNumber') QUERY NODESVALUE3('Price') QUERY NODESVALUE4('../Quantity') QUERY NODESVALUE5('Price\*../Quantity')

If, however, your document declares more than one namespace, and, especially where there would be a namespace collision without the use of the namespaces, it may be necessary to load the document in namespace-aware mode. This is done by specifying the \*NAMESPACEAWARE keyword in the XMLOPTIONS parameter of the XML\_QUERY activity. For example:

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:SalesOrder/@SONumber')

However, once the document is loaded in namespace-aware mode, the example queries shown up to this point will no longer function because now the namespace forms a part of the identification of nodes in the XML document.

There are a variety of ways to formulate your XPath expressions such that they will function in the way you require in namespace-aware mode and it is well beyond the scope of this document to attempt to cover all the options. However, here are a few examples that might help to get you started:

1. This example uses the *local-name* XPath built-in function to select nodes based on their *local name* (the node name WITHOUT the namespace prefix):

```
XML_QUERYXMLFILE('TUTorderNamespacePrefixed.xml')
XMLOPTIONS('*NAMESPACEAWARE')
QUERYNODES('//*[local-name() = "SalesOrder"]')
QUERYNODESVALUE1('@SONumber')
```

2.If multiple namespaces are used and 'SalesOrder' is ambiguous in this context, then you can extend the previous example to use the *namespace-uri* XPath built-in function:

```
XML_QUERYXMLFILE('TUTorderNamespacePrefixed.xml')
XMLOPTIONS('*NAMESPACEAWARE')
QUERYNODES('//*[local-name() = "SalesOrder"
and namespace-uri() = "urn:schemas-lansa-com:tutorder.dtd"]')
QUERYNODESVALUE1('@SONumber')
```

3.Alternatively, if you know that all instances of the XML document will use the same namespace prefixes (which, you should understand, is NOT strictly necessary for them to be valid, even though it may commonly be the case in

practice), then you can include the namespace prefixes in your XPath expressions (provided the document *is* loaded in namespace-aware mode):

```
XML_QUERYXMLFILE('TUTorderNamespacePrefixed.xml')
XMLOPTIONS('*NAMESPACEAWARE')
QUERYNODES('//tut:SalesOrder')
QUERYNODESVALUE1(@SONumber)
```

In summary, each of the following queries work successfully with the namespace prefixed version of the document as shown above, providing the document *is* loaded in namespace-aware mode AND providing the actual namespace prefix used in the XML document matches that assumed in the queries:

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:SalesOrder/@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:SalesOrder') QUERYNODESVALUE1('tut:Customer/@CustNumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:Customer[@CustNumber="543"]') QUERYNODESVALUE1('../@SONumber')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:SalesOrder[@SONumber="12345"]/tut:Line/tut:Part') QUERYNODESVALUE1('@PartNumber') QUERYNODESVALUE2('../tut:Quantity')

XML\_QUERYXMLFILE('TUTorderNamespacePrefixed.xml') XMLOPTIONS('\*NAMESPACEAWARE') QUERYNODES('//tut:Part[tut:Price>2.99]') QUERYNODESVALUE1('../../@SONumber') QUERYNODESVALUE2('@PartNumber') QUERYNODESVALUE3('tut:Price') QUERYNODESVALUE3('tut:Price') QUERYNODESVALUE5('tut:Price\*../tut:Quantity')
## XML\_VALIDATE

This activity validates an XML document file. If the document is found to be invalid, the activity ends in error – you may wish to use a CATCH directive in your Processing Sequence immediately after this activity in order to handle any errors that result.

At the minimum, the activity will verify (by its successful completion) that the XML document is well-formed. If the XML document contains or references a DTD or an XML schema and the DTD or schema can be accessed, then the document content will be validated against the DTD or schema.

#### **INPUT Parameters:**

XMLFILE : Required

This parameter specifies the path and name of the XML document file to be validated.

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

# XSL\_TRANSFORM

This activity applies a user-specified and provided XSL 1.0 stylesheet to the specified source file to create a transformed result file. For example, an XSL stylesheet might be used to transform an XML file to HTML or PDF for viewing or printing.

NOTE: The transformation is performed using LANSA Integrator's XMLParserService. This uses Apache Xalan-J as the XSL transformation engine. This implementation presently supports XSL version 1.0 compliant stylesheets.

## **INPUT Parameters:**

XSLSOURCE : Required

Specifies the path to the source XML file to be transformed.

For example:

c:\dir1\myfile.xml, OR

/mydir1/myfile.xml

**XSLTARGET** : Required

Specifies the path to the target file that will contain the result of the transformation.

For example:

c:\dir1\myfile.html, OR

/mydir1/myfile.html

**XSLSTYLESHEET** : Required

Specifies the path to the XSL stylesheet file that contains the XSL 1.0 instructions that determine how the transformation is performed.

For example:

C:\dir1\myxmlTOmyhtml.xslt, OR

/mydir1/myxmlTOmyhtml.xslt

#### **OUTPUT Parameters:**

There are no output parameters for this activity.

# **ZIP\_DIRECTORIES**

This Activity will zip a list of directories and place the contents of the directories, and any sub-directories into a zip archive. The directories to be included may be specified in a list. The name of the zip archive may be specified or a default name may be created.

The location for the zip archive must be specified. If the zip archive already exists, it may be overwritten or added to. If the zip archive does not exist it will be created.

Other Activities for zip processing:

**ZIP\_FILES** Zips a list of files and adds them to a zip archive.

**ZIP\_LIST** Lists the contents of an existing zip archive.

**ZIP\_UNZIP** Extracts a single file or the entire contents of a zip archive.

### **INPUT Parameters:**

#### **DIRECTORYLIST** : Required

This parameter should contain a list of directories. The list must contain at least one entry.

Each entry in the list should be the full directory path.

The contents of any sub-directories will also be included in the resulting zip archive. The base directory will be included with each entry added to the zip archive.

For example: Windows C:\mydirectory

IBM i /orders

## **ZIPARCHIVEDIRECTORY** : Required

This value should contain the full path to be used for the location of the zip archive file that will be created.

For example: Windows C:\zipdirectory

IBM i /savedorders

## **ZIPARCHIVENAME** : Optional

This optional value may contain the full name of the zip file to be created. If this value is not provided, a name will be generated using the System Setting value, Zip File Name Prefix, plus a unique number and a .zip extension.

For example: ZP147.zip

#### **CLEARARCHIVE** : Optional

This optional value may contain YES or NO. If this value is not provided it defaults to YES.

If YES, the zip archive file created is cleared before adding new entries.

If NO, the zip archive is not cleared before adding new entries. Therefore, if the zip archive already exists, the contents will be extended by the additional entries.

#### OUTPUT Parameters: ZIPARCHIVENAME

Upon successful completion, this parameter will contain the name of the zip archive actually used. This may be the same as the value input or may be the name generated by the activity.

### ZIPARCHIVEPATH

Upon successful completion, this parameter will contain the full path and name of the zip archive.

## **ZIP\_FILES**

This Activity will zip a list of files and place them in a zip archive. The files to be included may be specified in a file list. The name of the zip archive may be specified or a default name may be created.

The location for the zip archive must be specified. If the zip archive already exists, it may be overwritten or added to. If the zip archive does not exist it will be created.

Other Activities for zip processing:

TEXT_SUBSTITUTE	Zips a list of directories and adds them to a zip archive.
ZIP_LIST	Lists the contents of an existing zip archive.
ZIP_UNZIP	Extracts a single file or the entire contents of a zip archive.

## **INPUT Parameters:**

**FILELIST** : Required

This parameter should contain a list of files. The list must contain at least one entry.

Each entry in the list should be the full directory path and file name. For example: Windows C:\mydirectory\ord1.xml

IBM i /orders/OrdJan.csv

## **ZIPARCHIVEDIRECTORY** : Required

This value should contain the full path to be used for the location of the zip archive file that will be created.

For example: Windows C:\mydirectory

IBM i /orders

## **ZIPARCHIVENAME** : Optional

This value may contain the full name of the zip file to be created. If this value is not provided, a name will be generated using the system setting *ZIP archive file name prefix*, plus a unique number and a .zip extension.

For example: ZP147.zip **CLEARARCHIVE** : Optional

This value may contain YES or NO. The default is YES.

If YES, the zip archive file created is cleared before adding new entries. If NO, the zip archive is not cleared before adding new entries. Therefore, if the zip archive already exists, the files in the FILELIST will be added to the existing archive.

#### **OUTPUT Parameters: ZIPARCHIVENAME**

Upon successful completion, this parameter will contain the name of the zip archive actually used. This may be the same as the value input or may be the name generated by the activity.

### ZIPARCHIVEPATH

Upon successful completion, this parameter will contain the full path and name of the zip archive.

## ZIP\_LIST

This Activity will list the contents of a zip archive. The path and name of the zip archive must be specified.

The contents of the zip archive are output in a list.

This Activity uses the LANSA Integrator ZipService.

Other Activities for zip processing:

ZIP_FILES	Zips a list of files and adds them to a zip archive.
ZIP_DIRECTORIES	Zips a list of directories and adds them to a zip archive.
ZIP_UNZIP	Extracts a single file or the entire contents of a zip archive.

### **INPUT Parameters:**

**ZIPARCHIVE** : Required

This value should contain the full path and name of the zip archive whose contents are to be listed.

For example: Windows C:\savedirectory\orders.zip

IBM i /saveorders/sav.zip

# **OUTPUT Parameters:**

## FILELIST

This parameter will contain a list of files in the zip archive.

A maximum of 5,000 entries may be listed.

The format of each entry will depend upon how the zip archive was created. The base directory information will be included in each entry name if it was captured in the original zip.

For example, if the base directory was captured by the original zip an entry might be mydirectory\january\ord1.xml

If no directory was captured in the zip an entry will contain just the file name, such as ord1.xml

# ZIP\_UNZIP

This Activity will unzip the contents of a zip archive. The path and name of the zip archive must be specified. The contents of the zip archive are output in a list.

This Activity uses the LANSA Integrator ZipService.

Other Activities for zip processing:

ZIP_FILES	Zips a list of files and adds them to a zip archive.
ZIP_DIRECTORIES	Zips a list of directories and adds them to a zip archive.
ZIP_LIST	Lists the contents of an existing zip archive.

## **INPUT Parameters:**

### **ZIPARCHIVE** : Required

This value should contain the full path and name of the zip archive from which a single file or the complete contents are to be extracted.

For example: Windows C:\savedirectory\orders.zip

IBM i /saveorders/sav.zip

## **UNZIPTOPATH** : Required

This parameter must contain the destination directory into which the extracted files are placed.

For example: Windows c:\mydirectory

IBM i /inbound/orders

## **UNZIPFILENAME** : Optional

If this parameter contains a value, it should be the name of the single file which is to be extracted from the zip archive.

If this parameter does not contain a value, then the complete contents of the zip archive are extracted.

The extracted file/s will be placed in the directory specified in the UNZIPTOPATH parameter.

## **KEEPDIRSTRUCTURE** : Optional

This parameter may have a value of YES or NO. Default value is NO. If this parameter is YES, then the directory structure in the zip archive is preserved. This means that sub directories will be created in the

UNZIPTOPATH directory to match those in the zip archive.

**Note:** If a conflict of directory name occurs then the unzip will be abandoned.

If this parameter is NO, then all entries in the zip archive will be placed in the UNZIPTOPATH directory.

#### **OUTPUT Parameters:**

There are no output parameters.

# 2.2.18 Deprecated Activities

Some Activities from previous versions of LANSA Composer have been deprecated in this version.

In most cases, the deprecated Activities are renamed and a replacement Activity is provided with new or different features or behavior. New Processing Sequences should use the suggested alternate Activity.

Unless otherwise indicated, the deprecated Activities are still supplied, but are hidden for normal use such that they cannot be used in new or revised Processing Sequences. This preserves the behavior of existing Processing Sequences that used the original activity.

The deprecated Activities are:

- ZZARCHIVE\_FILE
- ZZARCHIVE\_FILE2
- ZZCALL\_FUNCTION
- ZZDELETE\_FILE
- ZZFTP\_INBOUND
- ZZMSG\_RECEIVE
- ZZTUT\_03\_AT

## ZZARCHIVE\_FILE

Original Activity Name: ARCHIVE\_FILE

Alternate Activity name: MOVE\_FILE

This activity was deprecated for the following reasons:

- Because it did not properly set the error result code when the file move fails.
- Because its parameters specify the file and path to be moved in a way that is inconsistent with other supplied activities

# ZZARCHIVE\_FILE2

Original Activity Name: ARCHIVE\_FILE2

Alternate Activity name: MOVE\_FILE

This activity was deprecated because it did not properly set the error result code when the file move fails.

### ZZCALL\_FUNCTION

Original Activity Name: CALL\_FUNCTION

Alternate Activity name: CALL\_FUNCTION

This activity was deprecated because the CALL\_FUNCTION activity was enhanced with the ability to call functions in a different LANSA system and/or partition. In doing so, the number of supported exchange parameters was reduced to seven (7). The deprecated version of the activity preserves support for nine (9) exchange parameter values for existing Processing Sequences that may use them (but does not provide the enhanced capabilities of the new CALL\_FUNCTION activity).

## ZZDELETE\_FILE

Original Activity Name: DELETE\_FILE

Alternate Activity name: DELETE\_FILE

This activity was deprecated because it did not properly set the error result code when the file delete fails.

## ZZFTP\_INBOUND

Original Activity Name: FTP\_INBOUND

Alternate Activity name: FTP\_INBOUND

This activity was deprecated because of incorrect behavior with respect to the use of the local target directory used by the activity. When this deprecated version was used with some FTP servers (especially Linux servers), then parts

of the remote directory path may have been added to the target directory path. This behavior was INCORRECT and has been corrected in the revised FTP\_INBOUND. However because existing implementations may have relied on this behavior, this deprecated version maintains the previous behavior.

## ZZMSG\_RECEIVE

Original Activity Name: MSG\_RECEIVE Alternate Activity name: MSG\_RECEIVE

This Activity was deprecated because it ends with an error status when no message is available to receive. The new version of the Activity ends with a normal (OK) status in this case, but a new output parameter contains a count of the messages received, allowing Processing Sequences that use it to better handle this case.

## ZZTUT\_03\_AT

Original Activity Name: TUT\_03\_AT

Alternate Activity name: NEXTNUMBER

This activity was used in the tutorial exercise LIC004 provided with LANSA Composer version 2.0 and earlier. The tutorials supplied with LANSA Composer version 3.0 and later use the supplied NEXTNUMBER activity instead.

# 2.3 Configurations

Configurations encapsulate the variable information required for supplied transport Activities and for database connections used with Transformation Maps. Typically Configurations encapsulate information such as:

- Source and target identification and addressing and
- Security credentials

By encapsulating the implementation-specific information in Configurations, LANSA Composer allows a single Activity and often a single Processing Sequence to be used with different inputs and outputs to serve different source and target - for example, for use with multiple trading partners.

Specific Configuration types support the supplied transport Activities. They are:

- 2.3.2 FTP Configuration
- 2.3.3 HTTP Configuration
- 2.3.4 SMTP Server Configuration
- 2.3.5 SMTP Mail Details Configuration
- 2.3.6 POP3 Mail Configuration
- 2.3.7 SMS Configuration
- 2.3.9 Messaging Configuration
- 2.3.10 LANSA System Configuration

In addition, the special type of 2.3.8 Database Configuration is provided to support encapsulating database details for databases used with Transformation Maps.

## 2.3.1 Work With Configurations

Using LANSA Composer you can create, maintain, and delete any number of Configurations according to your application's needs.

To work with Configurations, expand *Configurations* in the *Navigator* and click the Configuration type that you wish to work with. To find out how to locate and select Configurations to work with, refer to:

### Locating and selecting items in the Instance Lists

For information on common tasks associated with Configurations (such as creating, copying, deleting and printing definitions) refer to:

#### Working with definition items

For information on tasks associated specifically with Configurations, refer to the following headings:

Test a Configuration

For information specific to particular configuration types, refer to the following headings:

2.3.2 FTP Configuration

2.3.3 HTTP Configuration

- 2.3.4 SMTP Server Configuration
- 2.3.5 SMTP Mail Details Configuration

2.3.6 POP3 Mail Configuration

- 2.3.7 SMS Configuration
- 2.3.9 Messaging Configuration

2.3.8 Database Configuration

2.3.10 LANSA System Configuration

#### **Test a Configuration**

Some configurations will have a *Test* button. When clicked, it will run an activity that is used to check the validity of the configuration parameters. The parameters tested will differ depending on the type of configuration. At the end of the test, a processing sequence log will be shown to indicate whether connection has been successful and the error that has occurred.

## 2.3.2 FTP Configuration

The FTP Configuration may be used to set details of how you wish to send and retrieve files using file transfer protocols. Activities to make use of these configurations are provided. These include:

- FTP\_INBOUND
- FTP\_OUTBOUND
- FTP\_COMANDLIST

Information for an FTP Configuration is in three sections:

- Identification identifies the FTP Configuration and contains basic information.
- Connection information about the remote FTP Host and how to connect to that host.
- Details specific details relating to the type of FTP Configuration being set up.

## Identification

This identifies the FTP Configuration and contains basic information.

FTP configuration : Details (FTP_IN_AJAX_ORD-Ajax inbound orders)		
Details Notes Audit		
ID	FTP_IN_AJAX_ORD	
Description	Ajax inbound orders	
FTP type	Inbound	
Status	Active	

**ID** An identifier to uniquely identify this configuration

**Description** This should describe the configuration.

**FTP type** Inbound, Outbound or Command List.

**Status** Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status – supplied activities will end in error if they attempt to use a configuration that has inactive status.

# Connection

Following is information about the remote FTP Host and how to connect to that host.

Remote host Remote user Remote password		10.2.5.136 ftporder				
				Remote host	You must sp specified as name. The c selected) or TLS/SSL).	becify the name of the remote FTP server. It may be an IP address (nnn.nnn.nnn.nnn:port) or domain lefault value for the port is 21 or 22 (if SFTP is 990 (for an FTPS connection using implicit
				Remote user	Must be specified and contains the user profile to connect to the remote host.	
Remote password	The password that is used along with the remote user to connect to the remote host. This value is held encrypted. NOTE: the remote password will be ignored if you select SFTP and specify a Private key file. In this case the private and public keys are used for authentication instead of the password.					
Connection timeout	This field ca be establish timeout valu specified, th	an specify a timeout for the remote host connection ed, in milliseconds. For example, specify 5000 for ue of five seconds. If zero (or a negative value) is ten no connection timeout applies.				
Response timeout	This field ca wait, in mill value of five then no resp remote host here to prev communicat	an specify a timeout for the response (socket read) diseconds. For example, specify 5000 for a timeout e seconds. If zero (or a negative value) is specified, bonse timeout applies. If communications with the are unreliable, you should specify a timeout value ent the FTP operation apparently "hanging" if the tions link is broken in the middle of an operation.				

Use There are a number of protocols and security layers associated with what is collectively known as 'FTP'. This field allows you to specify the particular protocol and security layer that you wish to use. You may choose from the options that are further described below:

#### FTP: File transfer protocol

Standard FTP, when used alone, does not take any precautions to protect information transmitted during a session. This includes your username, password, and any files transmitted. In FTP all data is passed back and forth between the client and server without the use of encryption. This makes it possible for an eavesdropper to listen in and retrieve your confidential information including login details. This option may be most suitable when used over a controlled and secure private network and/or where the data being exchanged is not sensitive. It usually runs over TCP port 21.

#### FTPS: FTP over TLS/SSL (implicit) FTPS: FTP over TLS/SSL (explicit)

FTPS extends the File Transfer Protocol (FTP) using TLS/SSL to encrypt the control session and, if required, the data session. TLS (Transport Layer Security) and SSL (Secure Sockets Layer) are protocols that provide data encryption and authentication between applications in scenarios where that data is being sent across an insecure network.

In explicit mode, the connection is initially made using a plain socket in unsecured mode then the client and the server negotiates a change to a TLS/SSL connection using the AUTH subcommand. Implicit mode requires a secure control channel directly to the TLS/SSL FTP server using a port (usually 990) allocated specifically for such secure connections.

#### SFTP: SSH file transfer protocol

SFTP (SSH file transfer protocol) is a secure replacement for FTP that runs over a Secure Shell (SSH) session, usually on TCP port 22. SFTP is a binary protocol that encrypts both commands and data before sending over the one connection (separate command and data channels are not used). SFTP is becoming more popular, not least because it's supported by Linux and UNIX servers by default.

When you choose the SFTP (SSH file transfer protocol), the following.additional fields are presented:

Use protocol and security	SFTP: SSH file transfer protocol	
Private key file	/LANSA_Composer_licpgmlib/lic/ftpajaxorder.pem	
Private key password	*********	
Use SSH compression	No	

**Private key file**SFTP (SSH file transfer protocol) can use either user/password authentication or user and public key authentication. To use the latter, you usually will have to generate (or be provided with) two key files – a private key file (\*.pem) and a public key file (\*.pub). They should both be installed in the server file system in the SAME folder. You may also need to supply the public key file to your trading partner. The full path to the private key file should be entered in this field. You can click the prompt button to browse for a file on the LANSA Composer server. The FTP activities will expect to find a public key file with the same name and a ".pub" extension in the same folder. When using this mode of authentication the *Remote password* field is not used and will be ignored.

Private key<br/>passwordIf the private key file is encrypted, the password or passphrase<br/>to open the file should be entered in this field. This password<br/>is only used locally for this purpose – it is not sent to the<br/>remote server.

Use SSHThis field specifies whether or not the connection to the SFTPcompressionserver uses SSH compression.

When you choose the FTP or FTPS options, the following.additional fields are presented:

Use protocol and security	FTPS: FTP over	TLS/SSL (explicit)	•
Secure data channel	No		
Data link mode	Passive		
Binary mode	Yes		

Secure data channel when using FTPS, the control channel is always secure, but a secure data channel is not enabled by default. If the data being transmitted has already been encrypted then providing a further layer of encryption over the data channel simply imposes additional overhead. However, if the data being transferred has not already been encrypted, then you may select *Yes* in this field to specify that a

**Data** May be Passive or Active.

**link Passive** is the default.

**mode** Active causes port mode to be used by inbound list and get operations and by outbound put operations.

secure data channel should be requested.

**Binary** Indicates if the FTP transfer should be done in binary mode. This field does not apply and is not shown for a *Command List* type FTP configuration.

Note: The Test button can be used to check that the connection parameters are correct before saving. All parameters here are tested except *Data link mode* and *Binary mode*.

## Details

The details section will vary according to the FTP type specified in the Identification section.

#### Inbound

The Inbound FTP Configuration is for use in an Activity which connects to the remote host and uses the FTP GET command to retrieve files from that remote host and place the files on the local machine.

Local Directory Path	/AJAX/InOrder
Remote Directory Path	/JSM/BROKER/remote
Remote file list format	Auto-detect format
GET selection	GET files like
GET like pattern	*.ord
GET files using	Remote file name and extension
After GET action	Leave on remote host

Local directory path	This value must be specified and must contain the directory on the local machine where files which are retrieved from the remote host are placed. You can click the prompt button to browse for a folder location on the LANSA Composer server.
Remote directory path	The directory on the remote host from which files are retrieved. If not specified, files are retrieved from the user's default director
Remote file list format	The format of the file list which is retrieved from the remote host. Select from the dropdown list: Auto detect format (default), IBM i, Windows, Linux or NAME (use the FTP NLST subcommand).
GET selection	Select from the dropdown list: <b>GET all files</b> retrieves all the files in the remote directory. <b>GET files like</b> selectively retrieves files from the remote directory. A pattern specifying the type of files to be retrieved must be entered in the GET like pattern.
GET like pattern	This will be available if the <b>GET files like</b> is selected for the <i>GET selection</i> . It must contain a pattern to identify the files to be retrieved from the remote host.
	For example: <b>*</b> .ord will retrieve files with an extension of ord.
	ord* will retrieve files which start with ord.
GET files using	This option affects the local file name and extension that will be used for files received from the remote FTP server. You may wish to alter this option when, for example, the remote FTP server uses file names and/or extensions that do not comply with local file system naming rules.
	You can choose from the following options:
	Remote file name and extension

The same file name and extension used on the FTP server will be used as the local file name on the GET operation.

#### Generated file name, remote file extension

This causes the GET operation to receive the file using a generated local file name but with the same file extension (if any) used for

the file on the remote server.

#### Generated file name, no extension

This causes the GET operation to receive the file using a generated local file name with no file extension.

After Select from *Leave on remote host* or *Delete from remote host*.GET Leave on remote host leaves the files on the remote host after the

action GET action.

**Delete from remote host** deletes the files from the remote host directory after the GET action. You must have delete authority to the remote host directory to be able to use this action.

### Outbound

The outbound FTP Configuration is for use in an Activity which connects to the remote host and uses the FTP PUT command to place files from the local machine into a specific location on the remote host.

Local Directory Path	/Hammond/OutOrder
Remote Directory Path	/ftp/Orders/AJAX
PUT selection	Files with extension of
Extension	*.ord
Local Archive Path (OK)	/Hammond/FTPOut/OK
Local Archive Path (ER)	/Hammond/FTPOut/Error

Local	This value must contain the directory on the local machine which
directory	contains files which are to be placed onto the remote host. You
path	can click the prompt button to browse for a folder location on the LANSA Composer server.

**Remote** The directory on the remote host where files will be placed. If this directory is blank, files will be placed in the user's default directory.**path** 

**PUT** Select from the dropdown list:

**selection PUT all files** places all the files found in the Local Directory onto the remote host.

**Files with extension of** selects the files I the local directory which have a particular extension. The extension is specified in the

Extension.

Extension	This will be available if the "Files with extension of " is specified for the PUT selection. This should contain the extension of the files to be selected. The value is case insensitive and no "." is required.
Local archive path (OK)	This value is optional. It can contain a directory path on your machine. Outbound files which are successfully transported will be moved to this directory. You can click the prompt button to browse for a folder location on the LANSA Composer server.
Local archive path (ER)	This value is optional. It can contain a directory path on your machine. Outbound files which are encounter errors during the FTP processing will be moved to this directory. You can click the prompt button to browse for a folder location on the LANSA Composer server.

#### **Command List**

The command list FTP Configuration is for use in an Activity which connects to the remote host and executes the FTP commands which are specified in a file.

Command list file	/ftp/AdamsCmd 🕥			
Command list file	<b>I</b> This must be specified and contains the full path and name of a file which contains FTP commands.			
	The standard Activity executes the FTP commands using LANSA Integrator. For further information on the FTP commands and the keywords, refer to FTP commands.			
	The command list file should contain only one command per line. Comment lines should contain a # as the first character. The line terminator may be carriage return, line feed, new line, carriage return line feed or line feed carriage return.			

# **FTP commands**

These are some possible commands that can be used in FTP Command list:

Command	Des
CREATE DIR( <directory>)</directory>	Crea direa the 1 FTP A va mus supp the <dir< td=""></dir<>
DELETE DIR( <directory>)</directory>	Dele direo the r FTP A va mus supp the <dir< td=""></dir<>
DELETE FILE( <file>)</file>	Dele on tl remo host A va mus supr the <
RENAME FROM( <fromname>) TO(<toname>)</toname></fromname>	Rena file ( rema host Valu

	be si for <fro< th=""></fro<>
	<ton< th=""></ton<>
CHGDIR PATH( <new path="">)</new>	Char direc the r FTP A va mus prov the <ney< td=""></ney<>
BINARY	Cha bina mod
ASCII	Cha ASC
QUIT	End sess
GET FROM( <remotepath>) TO(<localpath>) CONVERT(*YES/*NO) FROM_ENCODING(<fromcode>) TO_ENCODING(<tocode>) RECORD_DELIMITER(&lt;*NONE/*LF/*CRLF) RECORD_SIZE(<size>) DATALINK(*PASV/*PORT)</size></tocode></fromcode></localpath></remotepath>	GE <sup>T</sup> from host plac loca mac Ref GE <sup>T</sup> com
PUT FROM( <remotepath>) TO(<localpath>) CONVERT(*YES/*NO) FROM_ENCODING(<fromcode>) TO_ENCODING(<tocode>) RECORD_DELIMITER(&lt;*NONE/*LF/*CRLF)</tocode></fromcode></localpath></remotepath>	PU from loca mac

RECORD_SIZE(*NOMAX/ <size>)</size>	onto
DATALINK(*PASV/*PORT)	remo
	host
	Refe
	PUT
	com

#### GET command

Keyword	Value	Notes
FROM	value	Mandatory. Remote file path. For example: /ftp/orders
ТО	value	Mandatory. Local file path. For example: /ftpin/orders
CONVERT	*NO *YES	Optional. Default is *NO (no conversion).
FROM_ENCODING	value	Optional. Only used if CONVERT(*YES) This is the encoding to convert the FROM file content to an intermediate Unicode string.
TO_ENCODING	value	Optional. Only used if CONVERT(*YES) This is the encoding to convert from the intermediate Unicode string to the target encoding.
RECORD_DELIMITER	*NONE *LF *CRLF	Optional. Only used if CONVERT(*YES). This specifies the end of record marker to be added to the end of each converted record read. Default is *NONE.

RECORD_SIZE	value	Optional. The read size used on the source stream. Default is 1024.
DATLINK	*PASV *PORT	Optional. *PORT - use port mode when getting the file. *PASV - use passive mode when getting the file. Default is *PASV.

# **PUT command**

alue	Mandatory. Local file path. For example: /outftp/orders
	1 1
alue	Mandatory. Remote file path. For example: /orders
NO YES	Optional. Default is *NO (no conversion).
alue	Optional. Only used if CONVERT(*YES) This is the encoding to convert the FROM file content to an intermediate Unicode string.
alue	Optional. Only used if CONVERT(*YES) This is the encoding to convert from the intermediate Unicode string to the target encoding.
	Ilue NO YES Ilue

RECORD_DELIMITER	*NONE *LF *CRLF	Optional. Only used if CONVERT(*YES). Default is *NONE. This specifies the end of record marker to be added to the end of each converted record read.
RECORD_SIZE	Value *NOMAX	The read size used on the source file. Optional. Defaults to 1024. *NOMAX sets the read size to the length of the source file.
DATLINK	*PASV *PORT	Optional. Defaults to *PASV. *PORT - use port mode when putting the file. *PASV - use passive mode when putting the file.

# 2.3.3 HTTP Configuration

The HTTP Configuration may be used to set details of how you wish to send and retrieve payloads using the http protocol. Activities to make use of these Configurations are provided. These are:

- HTTP\_INBOUND for inbound HTTP requests
- HTTP\_GET and HTTP\_POST for outbound requests

Information for an HTTP Configuration is in two parts:

- **Identification** Identifies the HTTP Configuration and contains basic information.
- **Details** Specific details relating to the type of HTTP Configuration being set up.

## Identification

This identifies the HTTP Configuration and contains basic information.

Details Details Ontes Audit			
ID	HTTPIN		
Description	Inbound HTTP		
HTTP Direction	Inbound		
Status	Active		

HTTPAn identifier to uniquely identify this ConfigurationConfiguration

ID

**Description** This should describe the Configuration.

HTTP<br/>directionInbound or OutboundConfiguration<br/>StatusActive or Inactive. Configurations cannot be used in a<br/>processing sequence while they are in inactive status –<br/>supplied activities will end in error if they attempt to use a<br/>configuration that has inactive status.

## Details

The details section will vary according to the HTTP direction specified in the

Identification section.

#### Inbound

The Inbound HTTP Configuration will set up details for a JSMDirect inbound HTTP. The information entered will create and maintain the function registry used by JSMDirect. This is the DC@W29 table on IBM i or the dc\_w29.txt on Windows.

Also the Processing Sequence which is to be launched is entered.

Processing Seq. ID	HTTP_INBOUND_SQ	 •
Application Name	TESTHTTP	
Local Host	*DEFAULT	
Handler Type	Function 💌	
Process Name	DXPROC02	
Function Name	DXHTFN1	
Partition	LIC	
Language	ENG	
OK HTTP response message	HTTP content received OK.	

Processing	This must contain the Identity of a Processing Sequence. This
Sequence ID	Processing Sequence should contain an Activity which will process the payload of the inbound HTTP transmission. A standard Activity HTTP_INBOUND is provided for this purpose.
Application Name	This is the application name which will be registered on the DC@W29/dc_w29.txt table. It identifies the application to be executed and is part of the URL. For example: http://lansa01:8080/cgi-bin/jsmdirect? ZZORDERS+ABC where the Application Name is ZZORDERS.
Local Host	This is the host name which will be registered on the DC@W29/dc_w29.txt table. It will default to *DEFAULT. It can contain a value of *DEFAULT or a HostName:Port
Handler Type	Select from Function or IBM i 3GL. The value selected for this Type will determine what next entry

fields are required.

OK HTTP<br/>responseThis optional entry may contain the text which is to be sent for<br/>an OK responsemessage

#### For a Handler Type of Function:

Process Name	This must be the process which contains the Function specified in the following parameter. This defaults to DXPROC02.
Function Name	This is the function that will be invoked by JSMDirect when an inbound HTTP transmission with the specified Application Name in the URL is received. This defaults to DXHTFN1 which is a supplied function in process DXPROC02. This supplied function will match the inbound URL to the Application Name on this Configuration and then launch the Processing Sequence specified on that Configuration. Refer to HTTP_INBOUND for a description of this function.
Partition	This must be the LANSA partition which contains the function to

- **Partition** This must be the LANSA partition which contains the function to be invoked.
- **Language** This must contain the LANSA execution language.

## For a Handler Type of IBM i 3GL:

**3GL** This should contain the program name when a Handler Type of**Program** IBM i 3GL is selected.

## Outbound

The outbound HTTP Configuration is for use in an Activity which sends HTTP requests to a remote HTTP server .

If applicable, the payload is specified as a directory path plus the file name of the content file. The content type may also need to be specified.

Remote Host	lansa01:4580	
URI	/cgi-bin/jsmdirect?testhttp	
Content File Directory	/httpout/	
Content File Name	/orders.txt	
Proxy Server		
Secure Protocol	No	
Authentication User		
Authentication password		
Wait for response	×.	
Content Type	*TEXTPLAIN	
Encoding		

Remote Host	This must contain the remote HTTP server and port. The format is host:port For example lansa01:8080 or 10.2.1.456:88	
	If the port is not supplied, it will default to 80.	
URI	This must contain the remainder of the URI to be used to connect to the remote server.	
Content File Directory	Enter the local directory path which contains the content file. You can click the prompt button to browse for a folder location on the LANSA Composer server.	
Content File Name	Enter the content file name. This file will be used as the payload of the HTTP transmission.	
<b>Proxy Server</b>	If a proxy server is being used, enter the proxy server name.	
Secure Protocol	Select Yes to use HTTPS (secure protocol) for the HTTP connection. Select No to use use standard HTTP protocol.	
Authentication User	If authentication is required, this contains the basic authentication user.	
Authentication password	If authentication is required, this contains the password which is used with the Authentication user.	
Wait for response	Select from Yes or No. Select yes if you want the HTTP connection to wait for a response.	

Response timeout	If you specified Yes for the Wait for response, then this field can specify a timeout for the response wait, in milliseconds. For example, specify 5000 for a timeout value of five seconds. If zero (or a negative value) is specified, then no response timeout applies.
Connection timeout	This field can specify a timeout for the host connection to be established, in milliseconds. For example, specify 5000 for a timeout value of five seconds. If zero (or a negative value) is specified, then no connection timeout applies.
Content Type	You may select the content type of the payload which will be sent:
*XML	application/xml
*TEXTXML	text/xml
*TEXTPLAIN	text/plain
*SOAP	application/soap+xml
*HTML	text/html
*XHTML	application/xhtml+xml
*CSV	application/comma-separated-values
*TEXTCSV	text/ x-comma-separated-values
*TSV	application/comma-separated-values
*TEXTTSV	text/ x-tab-separated-values
*X12	application/edi-x12
*EDIFACT	application/edifact
*STREAM	application/octet-stream
*PDF	application/pdf
*JSON	application/json
*EXCEL	application/vnd.ms-excel

*ZIP	application/zip
*GIF	image/gif
*PNG	image/png
*JPEG	image/jpeg
*MPEG	audio/mpeg
*MPEG4	video/mpeg4

Most often, however, you may leave the default Automatic setting, which will result in the HTTP\_POST activity setting an appropriate content type according to the file extension of the file being sent. (This is achieved by a lookup on the file *filetype.txt* in the *<system>* directory of the server's JSM instance directory tree.)

**Encoding** You may enter an encoding which will be used to convert the content to Unicode.

Note: A Test button can be used to check the validity of the parameters for Outbound HTTP Configuration. The parameters tested are: Remote Host, Proxy Server, Secure Protocol, Authentication User and Password, and Connection and Response timeout values, if applicable.
# 2.3.4 SMTP Server Configuration

The SMTP Server Configuration may be used to identify the SMTP server to be used when an email is sent using the SMTP protocol. Typically you will have a single SMTP Server Configuration that identifies your SMTP server.

A standard Activity named SEND\_MAIL is provided to send emails. This standard Activity uses the information in the SMTP Server Configuration to identify which email server is to be used.

Details 🖉 Attachments 🚺 Notes 📀 Audit		
ID	DEFAULT_SMTP	
Description	Default SMTP mail server	
Status	Active	
SMTP Server	10.2.20.55	
SMTP server user profile	MAILUSER	
SMTP server password	*****************	
TCP/IP port		
Use secure connection (SSL)	No	
Mail domain		
Character set for body & subject		
Signer		
Encryption algorithm	×	

**ID** An identifier to uniquely identify this Configuration

**Description** This should describe the Configuration.

- Status Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status supplied activities will end in error if they attempt to use a configuration that has inactive status.
- SMTP This is the IP address or domain name of the SMTP mail server. This value must be provided.
  SMTP This optional value is the user profile name used to login to the SMTP server.

profile

SMTP

server password	This optional value is the password used for login to the SMTP server.
TCP/IP port	This value is optional. The default value is port 25 (or 465 for a secure connection using Implicit SSL/TLS).
Use secure connection (SSL/TLS)	<ul> <li>Specifies whether to use a secure connection to the specified SMTP server. You can choose one of the following:</li> <li>Explicit SSL/TLS attempts to establish a secure connection in explicit mode. This initially connects in unsecured mode (typically on port 25) then negotiates a change to a SSL/TLS connection using the STARTTLS protocol command.</li> <li>Implicit SSL/TLS requires secure control channel directly to SSL/TLS SMTP server using a port (usually 465) allocated specifically for such secure connections.</li> <li>No will result in an unsecured connection being attempted (some servers may refuse or limit such a connection).</li> </ul>
Mail Domain	This value is optional and may contain the mail domain issued by the SMTP HELO subcommand.
Character set	This value is optional. It may contain the character set set encoding of the body text and subject.
Signer	This value is optional. It consists of 2 components, name:password.
	These values are used as keyword to identify entries in the LANSA Integrator SMTPMailService properties file.
	The X509 certificate file is obtained form the signer.certificate. {name} property.
	The PKCS8 keystore file is obtained from the signer.keystore. {name} property. The keystore password is used for the password.
	Refer to documentation of the SEND command of the SMTPMailService in the LANSA Integrator Guide for further information on the configuration and use of this value.
Encryption algorithm	This value is optional. It may contain *DES, *DES-EDE3, *RC2- 40, *RC2S-64 or *RC2-128.
	Refer to documentation of the SEND command of the

SMTPMailService in the LANSA Integrator Guide for further information on the configuration and use of this value.

Note: A Test button can be used to test the server configurations. The values tested are: SMTP Server, SMTP Server User, SMTP Server Password, TCP/IP port, Use secure connection (SSL) and Mail Domain.

# 2.3.5 SMTP Mail Details Configuration

The SMTP mail details Configuration may be used to set up information for an email that is to be sent using the SMTP mail protocol.

A standard SMTP email Activity named SMTP\_EMAIL is provided. This standard Activity uses the information in the SMTP Mail Details Configuration to provide basic details for the email to be sent. Also some of the details that are to be used on the email may be provided at execution time. Consequently a SMTP Mail Details Configuration may be set up with the basic, unchanging values and the remainder of the mail information provided at execution time.

For example you could set up a Mail Details Configuration for the standard order acknowledgement email. This standard acknowledgment email could then be sent with minor variations.

Details 🖉 Attachments 🚺 🕅	lotes 🕢 Audit
ID	EXAMPLE_SMTP
Description	Example SMTP mail details
Status	Inactive
To email address	tradingpartner@hisaddress.com
CC email address	
Blind copy email address	
From email address	fromaddresss@yourcompany.com.au
From email display name	FROMNAME
Email subject	Thank you for your order
Attachment location & file	
Zip file attachment	
Body text file (location & name)	
Content type	
Encoding	

The standard SMTP Activity uses LANSA Integrator.

**ID** An identifier to uniquely identify this email Configuration

**Description** This should describe the Configuration.

Status Active or Inactive. Configurations cannot be used in a

processing sequence while they are in inactive status – supplied activities will end in error if they attempt to use a configuration that has inactive status.

- **TO email** address This value must be provided. It should contain a valid email address, or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.
- **CC email address** This optional value specifies the CC (copy to) address. It can contain a single email address, or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.
- Blind copy<br/>email<br/>addressThis optional value specifies the BCC (blind copy) address. It<br/>can contain a single email address, or a list of email addresses<br/>separated by commas. Click the adjacent prompt button to enter<br/>or maintain the list of email addresses.
- From email<br/>addressThis value must be provided. It will default to the email addressaddressspecified in the system setting SMTP From e-mail address.
- From email<br/>displayThis value is optional. It may provide a value for the From Name<br/>as it is displayed to the email recipient. It will default to the<br/>email address specified in the system setting SMTP From e-mail<br/>display name.

# Email This value is optional. It may contain the standard subject for this email.

Attachment This optional value may contain the path and file name of a file that is to be attached to the email. The attachment may be added to the email unzip or zipped. It will be zipped and attached to the email if a value is specified in the Zip file attachment. You can click the prompt button to browse for a file name and location on the LANSA Composer server.

# Zip<br/>attachmentThis optional value may contain a path and file name. If a value<br/>is provided, then the attachment file specified in the preceding<br/>field will be zipped as per this value. It will then be attached to<br/>the email. You can click the prompt button to browse for a file

	name and location on the LANSA Composer server.
Body text file	This optional value may contain the path and file name of a file that contains the body text for the email. You can click the prompt button to browse for a file name and location on the LANSA Composer server.
Content type	This value is optional. It defaults to text/plain. Possible values are *PLAIN, *HTML or a specific value.
Encoding	This optional keyword may be used to specify the encoding that is to be applied when converting the body text file contents to Unicode before it is passed to JavaMail.

# 2.3.6 POP3 Mail Configuration

The POP3 mail Configuration may be used to specify details of your email server and how emails are to be retrieved from that email server using the POP3 protocol.

Two standard POP3 email Activities are provided. They are MAIL\_RECEIVE and MAIL\_RECEIVEALL. These Activities retrieve emails from the email server. Details of the email server can be set up in this Configuration

Details Notes Audit	
ID	EXAMPLE_POP3
Description	Example POP3 mail configuration
Status	Inactive
POP3 Mail server	123.123.123.123
POP3 MailServer user profile	Mailuser
POP3 MailServer password	*********************
POP3 port	
Public key name	
Public key password	
Save mail text	Yes
Save mail attachments	Yes
Save directory	/Mail
Remove from server after read	Yes

**ID** An identifier to uniquely identify this Configuration

**Description** This should describe the Configuration.

- Status Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status supplied activities will end in error if they attempt to use a configuration that has inactive status.
- **POP3 Mail**This is the IP address of your mail server. If this value is not<br/>provided on this Configuration, it must be specified in the<br/>LANSA Integrator POP3MailService properties file.
- **POP3** This is the user profile to be used to connect to your mail server.

MailServer user profile	If this value is not provided on this Configuration, it must be specified in the LANSA Integrator POP3MailService properties file.
POP3 MailServer password	This is the password which is used with the user profile to connect to your mail server. If this value is not provided on this Configuration, it must be specified in the LANSA Integrator POP3MailService properties file.
POP3 port	This value is optional. The default value is 110 (or 995 for a secure connection using Implicit SSL/TLS)
Use secure connection (SSL)	Specifies whether to use a secure connection to the specified POP3 mail server. You can choose one of the following: <b>Explicit SSL/TLS</b> attempts to establish a secure connection in explicit mode. This initially connects in unsecured mode (typically on port 110) then negotiates a change to a SSL/TLS connection using the STLS POP3 protocol command. <b>Implicit SSL/TLS</b> requires secure control channel directly to SSL/TLS POP3 mail server using a port (usually 995) allocated specifically for such secure connections. <b>No</b> will result in an unsecured connection being attempted (some servers may refuse or limit such a connection).
Public key name	This value is optional. It may be used to decrypt secure messages that have been encrypted with the recipient's public key. This value is used to access the appropriate private key store in the LANSA Integrator POP3 service properties.
Public key password	This value is optional. It is used with the Public key name as the password component.
Save mail text	Yes or No. If Yes is specified the text of the emails is to be saved to directory specified in Save directory.
Save mail attachments	Yes or No. If Yes is specified the email attachments are to be saved to directory specified in Save directory.
Save directory	This must be specified if the mail text or mail attachments are to be saved. You can click the prompt button to browse for a folder location on the LANSA Composer server.
Remove	Yes or No. If No is specified, then the emails that are read are

# **from server** left on the email server. This means that a subsequent connect to the email server and read will retrieve the same emails. It is usual to remove emails from the server after they have been read and the text and attachments saved.

Note: A Test button can be used to test the validity of the configuration parameters. The values tested are: POP3 Mail Server, POP3 MailServer user profile, POP3 MailServer password, POP3 port, Use secure connection (SSL), Public key name, Public key password.

# 2.3.7 SMS Configuration

The SMS Configuration may be used to specify the SMTP e-mail and other details necessary to send SMS messages via e-mail using a suitable SMS provider. It can also specify the destination number and SMS message text.

This configuration type is used by the SMS\_SEND activity. Refer to the description of that Activity for further information.

Details 🚺 Notes 🐼 Au	dit
ID	EXAMPLE_SMS
Description	Example SMS configuration
Status	Inactive 💌
SMTP Configuration ID	(none)
Or	
SMTP Server	123. 123. 123. 123
TCP/IP port	
Mail domain	
SMS user profile	SMSUser
SMS password	******************
Characterset for Body & Subject	
Encoding	
Mobile domain	streetdata.com.au
From email address	SMSOut@mycompany.com
To SMS number	04001234567
SMS Message	This SMS Message was sent from LANSA Composer

ID	An identifier to uniquely identify this Configuration
Description	This should describe the Configuration.
Status	Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status – supplied activities will end in error if they attempt to use a configuration that has inactive status.
SMTP	You can choose from the drop-down list an existing SMTP

Configuration ID	configuration which contains SMTP server, domain and port details that are used when sending e-mails to the SMS provider. SMTP configuration ID can also be set as the default SMTP configuration ID set in System Settings by choosing (Default) from the drop down. If you do not choose an existing SMTP configuration, then you may specify the SMTP server, domain and port details in the following fields.
SMTP server	If you do not choose an existing SMTP configuration, then this field must specify the IP address or domain name of the SMTP mail server used to send e-mails to the SMS provider.
TCP/IP port	If you do not choose an existing SMTP configuration, then this field can specify the port number of the SMTP mail server used to send e-mails to the SMS provider. If not specified, the default value used is port 25 (or 465 for a secure connection using implicit SSL/TLS).
Use secure connection (SSL)	If you do not choose an existing SMTP configuration, then this field can specify whether to use a secure connection to the specified SMTP server. You can choose one of the following: <b>Explicit SSL/TLS</b> attempts to establish a secure connection in explicit mode. This initially connects in unsecured mode (typically on port 25) then negotiates a change to a SSL/TLS connection using the STARTTLS protocol command. <b>Implicit SSL/TLS</b> requires secure control channel directly to SSL/TLS SMTP server using a port (usually 465) allocated specifically for such secure connections. <b>No</b> will result in an unsecured connection being attempted (some servers may refuse or limit such a connection).
Mail Domain	If you do not choose an existing SMTP configuration, then this field can specify the mail domain of the SMTP mail server used to send e-mails to the SMS provider.
SMS user profile	This is the user name provided by the SMS service provider. The SMS_SEND activity will place the user name and password, if specified, in the subject line of the generated email address in the form <i>user+password</i> . If your SMS provider uses a different e-mail format, the activity processor for the SMS_SEND activity may need to be modified to

accommodate it.

SMS password	This is the password for the user provided by the SMS service provider.
Character set	This value is optional. It may contain the character set encoding of the body text and subject.
Encoding	This value may be used to specify the encoding that is to be applied when converting the byte string.
Mobile	This is the mobile domain of the SMS provider.
domain	The SMS_SEND Activity will prepend the SMS number specified to this domain to form the email address to which to send the SMS message. For example if the SMS number is specified as '04001234567' and the mobile domain is 'streetdata.com.au' then the email will be sent to '04001234567@streetdata.com.au'.
	The resulting email address must conform to the form specified by your SMS provider. If your SMS provider uses a different e-mail format, the activity processor for the SMS_SEND activity may need to be modified to accommodate it.
From email address	This is the email address from where the SMS will be sent. Your SMS provider may send confirmations or other responses to this e-mail address or may require a particular email address be used for authentication and billing purposes. Refer to your SMS provider for information.
To SMS number	This is the SMS number to which the SMS will be sent. If not specified here, it must be specified in the parameters to the SMS_SEND Activity.
	Refer to the description of the Mobile domain field above for information on how this value is used to construct the destination email address.
SMS message	This value can contain the SMS message text to be sent. If not specified here, it must be specified in the parameters to the SMS_SEND Activity. Most SMS providers restrict the SMS message length to a maximum number of characters, for

example 160 characters. It is your responsibility to ensure the message text does not exceed the maximum length imposed by the service you are using.

# 2.3.8 Database Configuration

The database configuration is used to contain the details necessary for establishing a database connection for use with the 2.2.10 SQL Database Activities or with a Transformation Map.

In the case of a Transformation Map, if the map makes use of a database as input or output, its parameters will include one for the database connection. You provide the connection to the required database by specifying a database configuration name in this parameter.

Refer to 2.5 Transformation Maps for more information on defining and using Transformation Maps.

For more information about establishing database connections for use with LANSA Composer, refer to Connecting to Databases in Transformation Maps.

Details 📋 Notes 🕟 Audit	
ID	EXAMPLEDB_400
Description	Example System i database configuration
Status	Inactive
Database connection string	jdbc:as400://SYSNAME/LIBNAME;prompt=false;naming=sql;errors=full;date form
Database driver class	com.ibm.as400.access.AS400JDBCDriver
Database user	QUSER
Database password	*****
Commit automatically	Yes
Transaction isolation	Read uncommitted 💌

The information that must be supplied for a database configuration is described below:

IDAn identifier to uniquely identify this Configuration. To make use<br/>database Configuration, you will specify this name as the value fo<br/>DBCONFIG parameter of the SQL\_CONNECT activity or for the<br/>database connection parameter for a Transformation Map.

**Description** This should describe the Configuration.

**Status** Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status – supplied activities will

error if they attempt to use a configuration that has inactive status.

Database<br/>connectionSpecifies the JDBC connection string necessary to connect to the<br/>database from the environment in which the Processing Sequence<br/>The following is an example of a JDBC connection string:<br/>jdbc:as400://SYSNAME/LIBNAME;naming=sql;errors=f<br/>format=iso;true autocommit=true;translate binary=true

**Note:** When your transformation map uses database components that connect to IBM DB2 for i5/OS database tables, LANSA Composer will remove the library qualifier from the generated Java code when you prepare the transformation map. You must setup your database configuration so that the required files can be found at run-time. Usually this is done by specifying the library name as part of the URL in the JDBC connection string - in the example shown, LIBNAME represents the library name. For more information, refer to Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS .

Database driver class	Identifies the Java class name of the JDBC driver used with this connection. The following is an example class name for the driver commonly used to access IBM i databases. com.ibm.as400.access.AS400.IDBCDriver
Database user	If authentication is required, this contains the authentication user name.
Database password	If authentication is required, this contains the password which is used with the database user.
Commit automatically	This indicates if information should be automatically committed to the database using auto-commit support in the driver and/or database server. Depending on the particular database and driver, auto-commit in effect can make every SQL statement a transaction. Note that this can effectively defeat transaction control implemented at the application layer, for example, if the Use Transactions option is checked in the mapping tool. Commitment control can also be affected or controlled using options in the transformation map definition and in the JDBC

connection string. You would usually only use this setting when transaction control has not been implemented in the transformation map definition. These considerations, however, can vary according to the database system and the JDBC database driver.

# TransactionThe level of transaction isolation to be applied to the database<br/>connection.

**None** No transaction isolation applies. All uncommitted data is readable from any connection.

#### **Read uncommitted**

All uncommitted data is readable from any connection. This is the same as not having any isolation (None).

#### **Read committed**

This prevents dirty reads but does not prevent phantoms or non-repeatable reads. Using this isolation level, only data committed before the current transaction began will be available. Any dirty data or changes made by concurrent transactions will not be available.

This level is obviously more restrictive than the Read uncommitted.

#### **Repeatable read**

This prevents dirty and non-repeatable reads but does not prevent phantom rows. This means the probability of other transactions having to wait for this one are increased when compared to Read uncommitted and Read committed

This is more restrictive than Read committed.

#### Serializable

Serializable provides the highest transaction isolation. When a transaction is isolated at the Serializable level, only data committed before the transaction began is available. Neither dirty data nor concurrent transaction changes committed during transaction execution are available. This level emulates serial transaction execution, as transactions will effectively be executed one after another rather than concurrently.

# 2.3.9 Messaging Configuration

The Messaging Configuration is used to set details of how you wish to send and retrieve files using a supported messaging brokering system such as IBM MQ Series.

Activities to make use of this Configuration are provided. These are:

- MSG\_SEND
- MSG\_RECEIVE

**Note:** These activities make use of the JMSFileService of LANSA Integrator. This service uses the Java Message Service JMS 1.1 specification. Refer to http://java.sun.com/products/jms/docs.html

This service supports four message brokering systems: IBM MQ Series, ActiveMQ, SonicMQ and TibcoMQ. You need to separately install the message brokering system vendor's implementation of JMS 1.1. If you're using:

- MQSeries, then you require the IBM MQ Series Client for Java JMS jar files.
- SonicMQ, then you require the Sonic JMS jar file.
- ActiveMQ then you require the ActiveMQ JMS jar file.
- TibcoMQ then you require the TibcoMQ JMS jar file.

You will need to know how to use the enterprise messaging system selected for use with this JMSFileService. The setting up and configuring of either messaging system is beyond the scope of this guide.

Information for a Messaging Configuration is on two tabs:

- The Details tab contains information that mostly applies and is set when establishing a connection with the host and can contain information about the folders and file names that are used when sending and receiving files.
- The Message Properties tab can be used to specify message properties that are to be set when sending messages and where to receive message property information when receiving messages.

# Details

The Details command has 5 sections:

- **Identification**This identifies the Messaging Configuration and contains basic information.
- **Messaging System** –Information about the messaging system host and how to connect to that host.
- **Queue and Channel** Specifies the queue and channel that are to be used when sending and receiving messages through the message brokering system.
- **Send Details** Contains information used by the MSG\_SEND activity. Some of the details may be overridden by activity parameters.
- **Receive Details** Information used by the MSG\_RECEIVE activity. Some of the details may be overridden by activity parameters.

# Identification

This identifies the Messaging Configuration and contains basic information.

📄 Details 🛛 💊 Message pro	perties 🛛 🚺 Notes 🛛 🐼 Audit
ID	EXAMPLE_MSG
Description	Example messaging configuration
Status	Active

**ID** An identifier to uniquely identify this Configuration

**Description** This should describe the Configuration.

**Status** Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status – supplied activities will end in error if they attempt to use a configuration that has inactive status.

# **Messaging System**

This is used to specify information about the messaging system host and how to connect to that host.

Messaging system	MQSeries 💌
Messaging system host	localhost
Messaging system port	1414
Messaging system user	
Messaging system password	

Messaging<br/>systemChoose from the supported message brokering systems. The<br/>default when creating a new configuration is MQSeries.

Messaging<br/>systemName of the remote host. If the host is the same as the one you are<br/>connected to when running the activity, the default value of<br/>localhost should be valid. Otherwise, specify a host value. It may<br/>be specified as an IP address (nnn.nnn.nnn:port) or domain<br/>name (xxxxxxxxx:port).

Messaging<br/>systemOptionally specify the remote port. If not specified, the default<br/>value differs according to the specified message brokering system<br/>as follows:

IBM MQ Series: 1414 SonicMQ: 2506 ActiveMQ: 61616

TibcoMQ: 7500

Messaging<br/>system<br/>userOptionally specify a valid user id to connect to the host.Messaging<br/>systemThe password that is used along with the Remote user to connect<br/>to the host. This value is held encrypted.

```
password
```

# Queue and Channel

Specify the queue and channel that are to be used when sending and receiving messages through the message brokering system.

Queue name	USERAGENT.QUEUE
Channel	USERAGENT.CHANNEL
Queue manager	USERAGENT.QUEUE.MANAGER
Client ID	

Queue Name	Mandatory. The name of the queue.
Channel	Mandatory (IBM MQSeries only).
Queue Manager	Optional (IBM MQSeries only).
Client ID	Optional. Client identifier.

#### Send Details

Specify information relevant when sending messages.

Send folder		•
Send file name		
Message delivery	*NONPERSISTENT	
Message priority	4	
Message time to live	(In milliseconds, default is 0 = forever)	

Send	A fully qualified folder name where to pick the files to send. You
folder	can click the prompt button to browse for a folder location on the
	LANSA Composer server.

May be overridden using the parameters to the MSG\_SEND activity or by specifying a fully qualified file name.

- Send file<br/>nameName of file to be sent. When also specifying a Send folder, this file<br/>will have to exist in that folder. Otherwise, it can be fully qualified.May be overridden using the parameters to the MSG\_SEND<br/>activity.
- Message<br/>deliveryDetermines whether the message, once sent, will persist when the<br/>queue is shut down and restarted. Default is \*PERSISTENT.
- Message<br/>priorityDetermines the default priority of the sent message. Default is 4.priorityThis value may be overridden on a message basis using the<br/>JMSPriority message property.

Message<br/>time to<br/>liveDetermines the expiration of the message in milliseconds. The<br/>default value is 0 meaning it never expires.

# **Receive Details**

Specify relevant information when messages are –received.

Receive folder		
Receive file name		

**Receive** folder A fully qualified folder name where the received message will be placed. You can click the prompt button to browse for a folder location on the LANSA Composer server.

May be overridden using the parameters to the MSG\_RECEIVE activity.

ReceiveThe name given to the received file. When also specifying a Receivefilefolder, this file will placed in that folder. Otherwise, it can be fullynamequalified.

May be overridden using the parameters to the MSG\_RECEIVE activity.

# **Message Properties**

Use this tab to specify message properties to attach to sent messages and to specify where to receive message properties (if required) for received messages.

Send	File Paths				
Receive					
Message Properties Name	Value	Name	MY_PROPERTY		
JMSCORRELA JMSDELIVERY JMSDESTINAT JMSEXPIRATION JMSMESSAGEID JMSPRIORITY JMSREDELIVE JMSREPLYTO JMSTIMESTAMP IMSTYPE		Value Type String Integer Boolear	My property value	C Long Float Double	

There are two ways to send message properties:

- 1. In the configuration.
- 2. In a comma-separated file. The MSG\_SEND activity expects such a file to be formatted like this: type, name, value

#### **Type** String, length 1:

J – use this type only for JMSxxxxx properties. Failure to observe this rule will result in abnormal termination of the activity.

S –for a property of type string.

I – for a property of type integer. The valid range for integers is -2147483648 to 2147483647 (inclusive).

B – for a property of type bool.

H – for a property of type short. The valid range for short is -32,768 to 32,767 (inclusive).

L – for a property of type long. The valid range for long is -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (inclusive).

F – for a property of type float. The valid range for float is 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).

D – for a property of type double. The valid range for double is 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative).

Name String, length 32.

Value String, length 10000.

Likewise, the MSG\_RECEIVE activity can write the message properties contained in a received message to a file formatted as above.

Typically you might specify message properties and their values directly in the configuration for message properties that will have a static value (the same value for each message sent using this configuration). For message properties whose value varies for each sent message, you would use the comma-separated files to specify the message properties and their values.

When the same property is specified in both the configuration and in the comma-separated file, the latter takes precedence.

# **Message Properties File Paths**

Optionally specify the location of the comma-separated files where to get the message properties from (MSG\_SEND activity) and where to output the message properties (MSG\_RECEIVE activity).

- **Send** A fully qualified folder name where the MSG\_SEND activity will look for a file containing a set of properties to attach to the sent message.
- **Receive** A fully qualified folder name where the MSG\_RECEIVE activity will output the received message properties.

# **Message Properties**

Specify values for pre-defined (JMSFileService) message properties and/or to specify the names and values for user-defined message properties.

## Pre-defined message properties

The list is initially pre filled with message properties that are common across all message brokering systems when used through the JMSFileService of LANSA Integrator. The names of these properties start with JMS.

You cannot remove the pre-defined properties. However you can set the value to blank. The MSG\_SEND activity will not use properties with a blank value.

#### **User defined properties**

You can add and remove user-defined properties to be sent with messages when using this configuration.

To add a property, click the *Add* button and then type the property name and value and select the property type.

To remove a property, click the property name in the list and click the *Remove* button. Note that you cannot remove the pre-defined properties.

# 2.3.10 LANSA System Configuration

The LANSA system configuration is used to contain the details necessary to connect to another LANSA system to run a LANSA function or a LANSA Composer processing sequence in that system. It is used in conjunction with the CALL\_FUNCTION and COMPOSER\_RUN activities. For more information concerning the usage of the LANSA system configuration, you may wish to refer to:

## CALL\_FUNCTION

#### COMPOSER\_RUN

#### Appendix F. The LANSA Composer Request Server

Common information that must be supplied for any LANSA system configuration is described below:

IDAn identifier to uniquely identify this Configuration. To make<br/>use of a LANSA system configuration, you will specify this<br/>name as the value for the LANSACONFIG parameter of the<br/>CALL\_FUNCTION or COMPOSER\_RUN activities.

**Description** This should describe the Configuration.

**Status** Active or Inactive. Configurations cannot be used in a processing sequence while they are in inactive status – supplied activities will end in error if they attempt to use a configuration that has inactive status.

Additional information specific to this configuration type is in two parts.

- 1. LANSA Composer connects to the request server using:
- 2. The request server connects to the other LANSA system using:

## **1. LANSA Composer connects to the request server using:**

The first part of the LANSA system configuration specifies how LANSA Composer contacts the LANSA Composer request server that will execute the task (CALL\_FUNCTION or COMPOSER\_RUN) in the other LANSA (or LANSA Composer) system.

If LANSA Composer, the request server and the other LANSA system all reside on the same server, then you should simply select *On local system* in the *Connect to request server* drop-down. You do not need to enter any further information. If, however, the other LANSA system resides on a different server (\*), then you should select *On remote system* in the *Connect to request server* drop-down and complete the additional information as described below.

\* Note a LANSA Composer Remote Request Server license is required for the remote server in order to call a LANSA function or run a Processing Sequence on a different server system.

**Note:** At time of writing (with reference to LANSA V12SP1), connection from one IBM i server to another IBM i server is not supported. This restriction may be lifted in future LANSA versions.

**Note:** LANSA Composer must be installed on the remote server system in order to use this support, so that LANSA Composer on the source system can communicate with components of the LANSA Composer request server on the remote system. Additional software licenses may be required. Refer to Connecting to the LANSA Composer Request Server on a Remote System for further details.

1. LANSA Composer connects to the r	equest server using:
Connect to request server	On remote system
Keep connection active	Yes 💌
Server type	LANSA Composer server for IBM i
Server LU partner name	SERVER02
Server partition	LIC
Use Windows credentials	No
User	LICPGMLIB
Password	****
Client-server translation table	
Server-client translation table	
Server exceptional arguments	

#### Keep connection active

If your processing sequence will make multiple requests (CALL\_FUNCTION or COMPOSER\_RUN) using the same LANSA system configuration, then you may wish to specify *Yes* in this field in order to keep the connection active between the

	requests. If you specify <i>No</i> , then a new communications session to the remote system will be started and ended for each request.
Server type	Choose the server type (IBM i or Windows) that applies to the remote system. Note that the values that you must supply in the second configuration part vary according to your choice here.
Server LU partner name	Specifies the Server LU partner name that identifies the network name, port and other information necessary to connect to the target system.
	If LANSA Composer is running on a Windows server, this is the name as defined in the <i>LANSA Communications Administrator</i> .
	If LANSA Composer is running on an IBM i server, this is the name as defined in LANSA's <i>Work with Communications Routing Records</i> screen.
	Notes:
	1. The LANSA Communications routing record that defines the <i>Server LU partner name</i> for this purpose is defined on the LANSA Composer server. You cannot use the LANSA Communications Administrator on your client computer to define this.
	2. The connection information must refer to the LANSA Composer configuration on the target system that contains the LANSA Composer request server software
	3. The LANSA listener for that system needs to be active when LANSA Composer attempts to establish the connection.
Server partition	Specifies the partition name in the LANSA Composer configuration on the target system that contains the LANSA Composer request server software. In a standard LANSA Composer installation on the target system, the name will be LIC.
Use Windows credentials	Check this box to specify that LANSA Composer should use the credentials of the job running the LANSA Composer processing sequence to connect to the LANSA Composer request server on the target system. The target system must have been configured for Single Sign On and the user enrolled in order to use this option.

User and

password	Specifies the user name and password that LANSA Composer uses to connect to the LANSA Composer request server on the target system.
Client- server and Server- client translation tables	Specifies the Client-to-Server and Server-to-Client conversion table names to be used. No library name can be specified. If not specified, these values default to *JOB, meaning the translation table will be generated based on the client code page and the IBM i server job's CCSID. If this argument is *JOB then the Server-to-Client table must also be *IOB
	These values do not apply when the target system resides on a Windows server
Server exceptional arguments	This may be used to specify a string of further X_RUN parameter names and values required to connect to the LANSA Composer request server on the target system. You should refer to LANSA documentation of the X_RUN parameters for more information.

# 2. The request server connects to the other LANSA system using:

The second part of the LANSA system configuration specifies how the LANSA Composer request server connects to the other LANSA (or LANSA Composer) system.

The information required varies according to whether the other system resides on an IBM i server or a Windows server.

# LANSA Composer server for IBM i

The additional information necessary to connect to the other LANSA (or LANSA Composer) system for an IBM i server is:

	2. The request server connects to the	other LANSA s	system using:
1	LANSA program library name	LICPGMLIB	
1	LANSA partition name	LIC	
1	LANSA language code		

LANSA Specifies the name of the LANSA program library in the LANSA system that is to be used. For example: DC@PGMLIB name

LANSA	Specifies the name of the LANSA partition in the specified LANSA
partition name	system that is to be used. For example: SYS. The partition must be multi-lingual enabled.
LANSA language code	Optionally specifies the LANSA language code used to start LANSA or LANSA Composer in the target system. If not specified, or if the special value '*' is used, then the language code being used by LANSA Composer will be used to start LANSA in the target system

## LANSA Composer server for Windows

The additional information necessary to connect to the other LANSA (or LANSA Composer) system for a Windows server is described below. Most of these parameters correspond directly to LANSA X\_RUN parameters used to make the connection to the LANSA system. You should refer to LANSA documentation of the X\_RUN parameters for more information.

2. The request server connects to the	other LANSA system using:
LANSA system path	C:\Program Files\LANSA\X_WIN95
LANSA partition name	SYS
LANSA language code	ENG
User name	PCXUSER
Password	**********************
Database name	LX_LANSA
Database type	Microsoft SQL Server
Database user	
Database pswd	
LANSA system overrides	QUET=Y PRTR=*PATH

LANSA system path	Specifies the path to the X_WIN95 folder in the LANSA system to be used. For example: C:\Program Files\LANSA\X_WIN95\
LANSA partition name	Specifies the name of the LANSA partition in the specified LANSA system that is to be used. For example: SYS.
LANSA language code	Optionally specifies the LANSA language code used to start LANSA or LANSA Composer in the target system. If not specified, or if the special value '*' is used, then the language code

	being used by LANSA Composer will be used to start LANSA in the target system.
User name	Specifies the user name used to connect to the LANSA system. (This value corresponds to the X_RUN parameter USER=.)
Password	Specifies the password used to connect to the LANSA system. (This value corresponds to the X_RUN parameter PSPW=.)
Database name	This parameter identifies the user database used with the specified LANSA system. (This value corresponds to the X_RUN parameter DBID=.)
Database type	Specifies the type of database specified in the previous parameter. (This value corresponds to the X_RUN parameter DBUT=.)
Database user	Specifies the user name for the database login, if required. (This value corresponds to the X_RUN parameter DBUS=.)
Database password	Specifies the password for the database login, if required. (This value corresponds to the X_RUN parameter DBPW=.)
LANSA system overrides	This may be used to specify a string of further X_RUN parameter names and values required to connect to the specified LANSA system. You should refer to LANSA documentation of the X_RUN parameters for more information.

# 2.4 Trading Partners

Trading partners are an optional feature of LANSA Composer. They are particularly useful for organizing files into directories (typically for inbound and outbound transport), for sharing common Activities and Processing Sequences with different source and target locations by using variable inputs associated with the trading partner and for performing iterative processing.

Trading partners may be registered and transport Configurations and Transformation Maps linked to the trading partner. A directory structure may be used where each trading partner has a group of directories used for different types of files. This gives a structure to the data that makes for easier and better management of the multitude of files.

In addition, application-specific properties of trading partner may be globally defined and values for those properties entered for each trading partner. This makes the trading partner functionality extensible to accommodate custom requirements.

The trading partner information may be referenced at run-time (in a Processing Sequence) in two ways:

- When processing is being performed for a particular trading partner, the trading partner attributes and properties, the linked directories, Transformation Maps and/or Configurations can be accessed through built-in variables and used as variable inputs to the Activities and Transformation Maps to be executed.
- A Processing Sequence may contain a loop based on the built-in \*TRADINGPARTNERS variable to iteratively perform the same processing for all defined trading partners.

For more information on dynamically accessing trading partner variables in your processing sequence, refer to Using the Trading Partner Built-in Variables. The trading partner features may not be necessary or applicable in all cases.

#### Also see

- 2.4.1 Work With Trading Partners
- 2.4.2 Trading Partner Details
- 2.4.3 Trading Partner Groups
- 2.4.4 Trading Partner Properties
- 2.4.5 Trading Partner Data Interchange Attributes

2.4.6 Link Directories to a Trading Partner

2.4.7 Link Transformation Maps to a Trading Partner

2.4.8 Link Configurations to a Trading Partner

2.4.9 Outbound Numbering

# 2.4.1 Work With Trading Partners

You can create, maintain, and delete trading partner definitions according to your application's needs.

To work with trading partners, expand *Definitions* in the *Navigator* and click *Trading partners*. To find out how to locate and select trading partners to work with, refer to:

Locating and selecting items in the Instance Lists

For information on common tasks associated with trading partners (such as creating, copying, deleting and printing definitions) refer to:

Working with definition items

For information on tasks associated specifically with trading partners, refer to the following headings:

- 2.4.2 Trading Partner Details
- 2.4.3 Trading Partner Groups

2.4.4 Trading Partner Properties

2.4.5 Trading Partner Data Interchange Attributes

2.4.6 Link Directories to a Trading Partner

2.4.7 Link Transformation Maps to a Trading Partner

2.4.8 Link Configurations to a Trading Partner

2.4.9 Outbound Numbering

# 2.4.2 Trading Partner Details

The details for a trading partner definition consist of basic identification and status information and basic contact details:

🐨 Outbound numbering   💺 Cro Details 🛛 纋 Groups   💊 Prop	ss references   🖉 Attachments   🛄 Notes   📀 Audit   perties   🎭 Data interchange   💈 Linked directories   🍛 Linked maps   🗗 Linked configurations
Name	ATLAS
Description	Atlas Trading Company
Status	Active
Contact	Hercules
Address 1	4 Strength Way
Address 2	Weston
Address 3	
imail	
imail 2	
imail 3	
hone number	+612 98765432
Fax number	+1 123 456788
Receive processing sequence	TXDOC_RCV_TP
Send processing sequence	TXDOC_SEND_TP

Basic identification and status information for a trading partner definition is:

**Identifier** An identifier to uniquely identify this trading partner.

**Description** This should describe the trading partner definition.

Status Active or Inactive. Trading partner definitions cannot be used in a processing sequence while they are in inactive status, and they will not be included in processing sequence loops that specify the \*TRADINGPARTNERS built-in variable.

The trading partner definition can hold basic contact details. These details are largely for reference. LANSA Composer does not automatically make use of any of these fields. However your Processing Sequences can use them in any way you see fit by referencing the corresponding built-in variables. The contact information is:

**Contact** The name of a contact at the trading partner's organization.

**Address** The street address for the trading partner.

1, 2, 3

**Email,** These fields can specify email addresses used with this trading

**2,3** partner. Each field can specify a single email address or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.

Phone The telephone number for the trading partner.Fax The fax number for the trading partner.

example of doing this).

The trading partner definition can specify specific processing sequences associated with the trading partner in connection with LANSA Composer's transaction document processing and framework:

Receive Specifies the name of a processing sequence that is used to **processing** receive inbound transaction documents for this trading partner. This processing sequence, if specified, will be called by the sequence supplied TXDOC RCV processing sequence to perform inbound processing specific to this trading partner. You can use it in your own processing sequences too, by referencing the associated trading partner built-in variable (refer to TXDOC RCV for an example of doing this). Send Specifies the name of a processing sequence that is used to send processing outbound transaction documents for this trading partner. This sequence processing sequence, if specified, will be called by the supplied TXDOC\_SEND processing sequence to perform outbound processing specific to this trading partner. You can use it in your own processing sequences too, by referencing the associated

trading partner built-in variable (refer to TXDOC SEND for an
#### 2.4.3 Trading Partner Groups

This tab lists the available Trading Partner groups – the groups to which this Trading Partner has been assigned have a checkmark next to their name. Trading Partner groups are used to group related Trading Partners so they can be easily located. You can define your own groups in addition to or instead of the groups supplied with LANSA Composer.

Linked maps     Linked maps     Details     Grou	Linked configurations   💟 Notes   🕟 Audit   ps   💊 Properties   🍕 Data interchange   💈	Linked directories
Name CUSTOMER DIVISIONS GOVERNMENT REGULATORY SUPPLIER UNCLASSIFIED	Description Customers Trading divisiions Government bodies Regulatory bodies Suppliers Unclassified trading partners	Refresh

Here you can assign Trading Partners to Trading Partner groups or remove them from Trading Partner groups.

To add a Trading Partner to a Trading Partner group, check the box next to the desired group. To remove a Trading Partner from a Trading Partner group, uncheck the box. Remember to click *Save* before switching elsewhere in LANSA Composer.

### **2.4.4 Trading Partner Properties**

Installation-specific properties for Trading Partners can be defined in Code Maintenance. If your installation has defined custom properties for Trading Partners, then you can enter values for those properties for each Trading Partner.

To review or change the property values for a Trading Partner, select the required trading partner in the trading partners list and then click the *Properties* tab in command handler area.

Linked configurations     Linked configurations     Details     Details     Details	🛛 🚱 Audit 🔤 🚱 Data interchange 🛛 🗿 Linked directories 🛛 🏠 Linked maps 🖉
ABN Number	Save
Credit Rating	Refresh

To modify a Property value, type the new value. Remember to click *Save* when complete to save your changes.

For information about accessing property values for a trading partner in a processing sequence, refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers.

#### 2.4.5 Trading Partner Data Interchange Attributes

LANSA Composer defines and stores a set of data interchange attributes for each Trading Partner. These are defined to accommodate some of the attributes commonly used in EDI and other forms of data interchange.

While LANSA Composer pre-defines fields to hold these attributes, the program performs only limited validation or enforcement of their entry and values. To permit maximum implementation flexibility, LANSA Composer leaves the use and values of these attributes largely to you according to your own circumstances and conventions.

Some of these values are, however, important for the operation of the transaction document framework as supplied. Refer to Trading Partner Data Interchange Attributes for more information.

To review or change the data interchange attributes for a Trading Partner, select the required trading partner in the trading partners list and then click the *Data interchange* tab in the command handler area.

Linked configurations   📅 Outbou	nd numbering   🖶 Cross references   🖉 Attachments   🛄 Notes   🥃 ies 🧠 Data interchange   🗿 Linked directories   分 Linked maps	) Audit
Communication agreement		Save
Recipients password qualifier		🔄 Refresh
Recipients password		
Inbound archive file prefix	idoc	
Outbound archive file prefix	odoc	
Inbound receiver ID	01 - 12345678	
Inbound receiver routing address		
Inbound sender ID	01 - 601943181	
Inbound sender routing address		
Outbound application sender	TXDOC01	
Outbound application receiver	TXDOC01	
Outbound receiver ID	ZZ - TXDOC01	
Outbound receiver routing address		
Outbound sender ID	ZZ - TXDOC01	
Outbound sender routing address		
EDI validation	Errors	
EDI split maximum transactions	000	
EDI split ignore carriage returns	No	
Overdue FA time interval (hhh:mm:ss)	00:00:00	
Exchange trading partner	TXDOC_EXCHANGE	
Exchange transaction cost	.01000	
Transaction cost unit of measure	AUD	

To modify a value, type the new value. Remember to click *Save* when complete to save your changes.

For information about accessing data interchange attribute values for a trading partner in a processing sequence, refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers.

## 2.4.6 Link Directories to a Trading Partner

To review or change the directories linked to a trading partner definition, select the required trading partner in the trading partners list and then click the *Linked Directories* tab in the command handler area.



You can add, remove or modify linked directories:

- To add a linked directory type, press the *Add* button. Refer to Add Linked Directories for more information.
- To remove a linked directory type for this trading partner, highlight the line to be removed and press the *Remove* button.
- Select the *Directory Path* column to specify or modify the path for the directory that applies to this trading partner.

Remember to *Save* your details before switching elsewhere in LANSA Composer.

Refer to Code Maintenance for information on defining directory types to be used with LANSA Composer.

For information about accessing linked directory names for a trading partner in a processing sequence, refer to Trading Partner (\*TRADINGPARTNER) Builtin Variable Qualifiers.

# **Add Linked Directories**

When creating or copying a Trading Partner or when adding Linked Directories to an existing Trading Partner, LANSA Composer displays the *Add Linked Directories* window. This window shows the directory types that are not already linked to the Trading Partner and:

- Automatically generates suggested directory paths according to system settings
- Allows selection of multiple directory types to be added in a single operation
- Can optionally create the generated or specified directory on the server

🗿 Т	Trading Partner EXAMPLE_TP - Add Linked Directories						
LANSA Composer can link directories of the types shown below for this trading partner. If necessary, LANSA Composer can also create the specified directories. Check the directory types that you want LANSA Composer to link to this trading partner. Type the directory path to use for each selected directory type.							
	Directory Type Directory Path						
1		IDOC_ARCHIVE	DOC_ARCHIVE Inbound transaction documen /LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/IDOC_ARCHIVE/				
2		IDOC_INERROR	inbound transaction documer /LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/IDOC_INERROR/				
3		IDOC_PROCESS	Inbound transaction documen	/LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/IDOC_PROCESS/			
4		IDOC_RECEIVE	Inbound transaction documen	/LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/IDOC_RECEIVE/			
5		ODOC_ARCHIVE	Outbound transaction docume	/LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/0D0C_ARCHIVE/			
6		ODOC_INERROR	Outbound transaction docume	/LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/0D0C_INERROR/			
7		ODOC_PREPARE	Outbound transaction docume	/LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/0D0C_PREPARE/			
8		ODOC_SEND	Outbound transaction docume /LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/ODOC_SEND/				
9	9 TUT Tutorial /LANSA_Composer_licpgmlib/lc3/EXAMPLE_TP/TUT/						
<b>I</b>	✓ Create directories if they do not exist     OK   Cancel						

Click the checkbox nex to each directory type that you wish to add for this Trading Partner. Type the directory path if you do not wish to accept the suggested directory path. Click OK when complete. LANSA Composer will link the selected directory types to the Trading Partner. LANSA Composer may also create the directories if they do not exist, according to the setting of the checkbox near the bottom of the window.

#### 2.4.7 Link Transformation Maps to a Trading Partner

To review or change the Transformation Maps linked to a trading partner definition, select the required trading partner in the trading partners list and then click the *Linked Maps* tab in the command handler area.

E	Notes 🛛 🕟 Au	udit								
	Details 🛛 🦢 G	roups	📎 Prope	erties 🛛 🍳	Data interchange	🗿 🗿 Linked directories	maps 🛛 🛃 Li	nked configuration	ns 🛛 📅 Outbo	und numbering
Γ	Map type	Seq	Ack	P/T	Map ID	Description	Map direction	Map standard	Trans type	I Save
1	nspecified typ	5	Yes	Production	1850_4010	EDI 850 into Staging Tables	Inbound	EDI X12 004010	850	
2	nspecified typ	10	No	Production	O997_4010	Staging Tables to EDI 997	Outbound	EDI X12 004010	997	Refresh
3	nspecified typ	15	No	Production	ITUT_ORDERS	Inbound XML to tutorial order databa	Inbound	ML TUTORDERS	ORDERS	
4	nspecified typ	20	No	Production	OTUT_ORDERS	Tutorial order database to outbound	Outbound	ML TUTORDERS	TUTORD	<u>A</u> dd
										<u>R</u> emove

You can add, remove or modify linked Transformation Maps:

• To add (or replace) a linked Transformation Map, press the *Add* button. Then select the Map Type and Transformation Map ID from the pop-up list and click *OK*.

**Note:** You can add any number of linked Transformation Maps for any one defined map type. If you add more than one map of the same type, then the value entered in the Sequence column will affect the evaluation order when you refer to Trading Partner linked Transformation Maps in a Processing Sequence using the Trading Partner built-in variables or using the FIND\_TPMAP activity.

• To remove a linked Transformation Map for this trading partner, select the line to be removed and then press the *Remove* button.

Remember to Save before switching elsewhere in LANSA Composer.

Refer to Code Maintenance for information on defining transformation map types to be used with LANSA Composer.

For information about accessing linked TransformationMaps for a trading partner in a processing sequence, refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers.

**Note:** To execute a Transformation Map linked to a trading partner in a processing sequence you must use the supplied TRANSFORM activity. Refer to TRANSFORM for information on this activity.

## 2.4.8 Link Configurations to a Trading Partner

To review or change the Configurations linked to a trading partner definition, select the required trading partner in the trading partners list and then click the *Linked Configurations* tab in the command handler area.

D	etails   🖉	9 Groups   💊 Prope	erties 🛛 💁 Data inte	rchange 🛛 🗿 Linked directories 🛛 🔗 Linked maps 🗍 🚰 Linked configurati	ons [ 🛄 N	lotes 🛛 🕟 Audit 🛛
2.1.52		Configuration Type	Configuration ID	Description	Status	E Save
1	FTPIN	FTP Inbound	DEM_FTP_IN	Demo FTP Inbound	Active	Back
2	FTPOUT	FTP Outbound	DEM_FTP_OUT	Demo FTP Outbound	Active	Refresh
						<u>A</u> dd)
						<u>R</u> emove

You can add, remove or modify linked Configurations:

• To add (or replace) a linked Configuration type, press the *Add* button. Then select the Configuration Type and Configuration ID from the pop-up list and click *OK*.

**Note:** You can only add one linked configuration for each configuration type. If you select a configuration type that is already linked to the trading partner, your selection simply replaces the existing instance for that configuration type.

• To remove a linked Configuration Type for this trading partner, select the line to be removed and then press the *Remove* button.

Remember to Save before switching elsewhere in LANSA Composer.

For information about accessing linked Configurations for a trading partner in a processing sequence, refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers.

# 2.4.9 Outbound Numbering

To review or change the outbound numbering for a trading partner definition, select the required trading partner in the trading partners list and then click the *Outbound Numbering* tab in the command handler area.

Outbo	ound numbering doma	in By transaction type for	the trading partner		Save
_					Refresh
		Next	Next	Next	
	Transaction Type	Number	Number	Number	<u>A</u> dd
1	997	9	9	9	Remove
2	TUTORD	8	8	8	

Outbound numbering is used to allocate control numbers for outbound transaction documents. As supplied, LANSA Composer's transaction document processing framework uses outbound numbering as follows:

- The TXDOC\_ALLOCCTRL activity uses the outbound numbering values for the specified trading partner to allocate the requested control numbers.
- Supplied export processors for selected EDI transactions (currently just X12 997 acknowledgement) use the outbound numbering values for the trading partner to allocate interchange, group and transaction control numbers for the outbound transaction(s).

The API for allocating the control numbers is available for custom export processors to use.

• When a pending outbound transaction document is registered using the TXDOC\_REGOUTBND or TXDOC\_REGOUTX12 activities or the supplied APIs, LANSA Composer uses the outbound numbering values for the trading partner to allocate the necessary control numbers.

Using the *Outbound numbering domain* drop-down list, you can choose, by trading partner, to allocate outbound transaction document control numbers in one of two ways:

• **By trading partner (all transactions)**: control numbers for this trading partner are allocated from the one series irrespective of the transaction type.

When you choose this option, the grid underneath changes to have a single line for transaction type *All* and you cannot add or remove entries.

• By transaction type for the trading partner: control numbers for this

trading partner are allocated from a separate series for each transaction type for which control numbers are requested.

When you choose this option, the grid underneath changes to show a line for each transaction type for which outbound control numbers have previously been requested (there may be none).

You may use the Add and Remove buttons to explicitly add entries for specified transaction types. Note, however, that it is only necessary to do so if you wish the numbering to begin at a value other than 1 (one). Otherwise LANSA Composer will automatically add entries when outbound control numbers are requested for a new transaction type.

The grid allows you to alter the next interchange, group and transaction number values one each line. Click in the required box and type the new value to change the values.

If you made any changes, remember to click *Save* before switching elsewhere in LANSA Composer.

**Note:** You should be fully aware of any auditing requirements or the requirements of your trading agreements with your trading partners before altering the next outbound control numbers.

## 2.5 Transformation Maps

Transformation Maps define how to transform or map data between disparate formats including:

- XML documents
- EDI documents
- Microsoft Excel 2007 (XLSX) files
- Text files (for example, comma-separated values)
- Web service functions
- A wide range of databases, including IBM DB2 Universal Database for IBM i

The usual procedure for creating and using a Transformation Map is as follows:

- 1. Create a Transformation Map definition in LANSA Composer as described in *Create or Copy a Transformation Map* in 2.5.2 Work With Transformation Maps.
- 2. Edit the Transformation Map in the mapping tool to perform the mapping required for your application as described in Edit Transformation Map.
- 3. Prepare the Transformation Map for use in a processing sequence as described in Prepare Transformation Map.

You can then use the Transformation Map in processing sequences.

#### Also see

- 2.5.1 Understand Transformation Maps and the Mapping Tool
- 2.5.2 Work With Transformation Maps
- 2.5.3 Transformation Map Details
- 2.5.5 Edit And Prepare Transformation Map

## 2.5.1 Understand Transformation Maps and the Mapping Tool

LANSA Composer includes Altova MapForce, a powerful visual mapping tool, for defining these transformations. The Transformation Maps created with the visual mapping tool can be directly integrated into your business processes along with transport and other activities.

Here's how LANSA Composer and Altova MapForce combine to define and use Transformation Map:

- When you *edit* your Transformation Maps, LANSA Composer will start MapForce to provide the visual mapping tools. You use the tools in MapForce to visually define source and target components and connections between the data items in each.
- When you *prepare* your Transformation Maps, LANSA Composer silently and invisibly invokes MapForce to generate the Java code for the map, and then LANSA Composer compiles the code and installs it ready for use.
- When a Processing Sequence *executes* your Transformation Maps, it simply invokes the generated and compiled Java code via a LANSA Integrator service provided for the purpose. Neither Altova MapForce, nor any other Altova software need be installed on the server computer at run-time.

**Note:** LANSA Composer uses only specific MapForce functionality and user interface options. Please refer to Unsupported MapForce Functionality for a list of unsupported functionality. Should you use an unsupported function, Transformation Map definitions that make use of these functions may not integrate correctly with LANSA Composer.

#### 2.5.2 Work With Transformation Maps

You can create and maintain as many Transformation Maps as your applications need.

To work with Transformation Maps, expand *Definitions* in the *Navigator* and select *Transformation maps*. To find out how to locate and select Transformation Maps to work with, refer to:

Locating and selecting items in the Instance Lists

For information on common tasks associated with Transformation Maps (such as creating, copying, deleting and printing definitions) refer to:

Working with definition items

For information on tasks associated specifically with Transformation Maps, refer to the following headings:

2.5.5 Edit And Prepare Transformation Map
Run a Transformation Map
Review Transformation Map Parameters
Display Catalogued Transformation map Information
Work with Transformation Map Version History
Display the Run History of a Transformation Map
2.5.3 Transformation Map Details
2.5.4 Transformation Map Data Interchange Attributes

## **Run a Transformation Map**

You can run an existing Transformation Map directly, without having to first create a Processing Sequence containing the Transformation Map.

To do so, select the required item in the Transformation Maps list. Details of the selected Transformation Map will be displayed. Click the *Run* button to run the Transformation Map. LANSA Composer will display a window like the *Run Processing Sequence* window in which you can enter the input parameter values for the Transformation Map.

Refer to Run a Processing Sequence from LANSA Composer.for further information on using the *Run Processing Sequence* window.

Running a Transformation Map in this way is very much like running a Processing Sequence containing just that Transformation Map. LANSA Composer will automatically log the input and output parameter values in the Processing Sequence log (subject to the logging level currently in effect).

# **Review Transformation Map Parameters**

To review the parameters for an existing Transformation Map, select the required item in the Transformation Maps list and select the *Parameters* tab. A list of the parameters expected for the transformation map will be displayed.

	Attachments   🛄 No Details   💁 Data inte	tes   🌝 Audit   rchange   🙀 Parameters   🛠 Catalogued   🔂 Run history   [	🔁 Version history    🍋 Cross i	references
	Parameter name	Parameter description	Parameter class	Refresh
1	DEM_ORDER_DB2_1	TU torder2 source filename	File path	
2	DEM_ORDER_DB2_2	Doc number source parameter	General	
3	DEM_ORDER_DB2_S	TXDOC_DATABASE target connection	Database configuration	

The parameter information is derived from the map definition once you have fully prepared the Transformation Map. It is provided for information and can only be altered by editing the map. When you include the Transformation Map in a processing sequence, you will need to provide values for each Transformation Map parameter.

Parameters provide the variable information necessary for the Transformation Map to complete its work. They also provide the communication between activities and transformations that are orchestrated together in a single processing sequence. Thus, the output (outbound parameters) from an activity might provide the input (inbound parameters) for a Transformation Map.

LANSA Composer derives the parameter information from the generated source code for the map. Consequently the parameter names and parameter descriptions shown may not be very meaningful. It is a good idea to define notes for the Transformation Map to clearly describe the expected input parameters, and to do this immediately after defining the Transformation Map, while its definition is fresh in your mind.

# **Display Catalogued Transformation map Information**

When a Transformation Map is prepared, LANSA Composer collects and catalogues a selection of information about the map. This includes:

- The date and time and user name when the map was prepared;
- LANSA Composer, Java and Altova MapForce version information relating to the Prepared map;
- File size and checksum information pertaining to the map definition file(s) and the generated Java executable file;

- The map definition file(s) and the generated Java executable file at Prepare time are saved in the LANSA Composer database (and can be restored if required);
- Selected mapping information collected from the Altova MapForce map definition, including the components used and the database table selections for database components.

To review the information catalogued for an existing Transformation Map, select the required item in the Transformation Maps list and select the *Catalogued* tab.

📄 Details 🛛 🔍 Data interchange	😫 Paramet	ers 🛛 🛠 Catalogued 🛛 🔂 Run history 🛛 🖶 Version history 🖓 🔍 Wher	e used 🛛 🚺 Notes
	Transforn verify the	nation map was successfully prepared by user LICPGMLIB. Click Check to current status of the files.	<u>C</u> heck
Prepared	2011-10	-27 15:01:20 by user LICPGMLIB	
Prepared by	LANSA C	Composer version 3.5.348	
Prepared using	MapFord	e version 2012	
Prepared to target	Java 1.5		
Map definition last saved with	MapFord	e version 2010	
Mapping		defaultmap1	
<ul> <li>Input components</li> </ul>			
Input		DocNumber	
		document	
<ul> <li>Output components</li> </ul>			
🖃 📙 Database		structure	
Data source name		TXDOC_DATABASE	
Strip schema names	(before pr	1	
Database type	(m. 1.4	IBM iSeries	
+ Database connection	n (* while e		
<ul> <li>Database selections</li> </ul>			
Table		TUTORDH	
Table		TITOPDI	
TODIC		TOTORDE	

You can press *Check* to compare the file sizes and checksums of the current Transformation Map definition and executable files against those catalogued when the map was last prepared.

#### Work with Transformation Map Version History

LANSA Composer can store multiple versions of the definition of a Transformation Map. This permits the flexibility to restore and run an earlier version of the Transformation Map if required.

When a Transformation Map is successfully prepared, the prior (prepared) version is archived (subject to the corresponding system setting). The archived version contains the complete definition and executable form of the Transformation Map, but does not include the attachments or notes.

To review the the available prior versions for an existing Transformation Map, select the required item in the Transformation Maps list and select the *Version History* tab. You can delete or restore the archived versions using the buttons provided.

# **Display the Run History of a Transformation Map**

To display available Run History for a Transformation Map, select the required item in the Transformation Maps list and select the *Run History* tab. A list of processing sequence runs (including direct runs of the Transformation Map) that used the selected Transformation Map will be displayed. You can double-click an item in the list to view the Processing Sequence Log.

Select an item and click the *View* button to display the Processing Sequence log or the *Print* button to print the Processing Sequence log for the run.

If a prior Processing Sequence run has ended in error, you may be able to use the *Restart* button to restart it from the point of failure. Refer to Restart a Processing Sequence Run for more information.

**Note:** The completeness of the Run History is subject to the logging level that was in force for each run. Only at more detailed levels of logging is the log information sufficient to identify each and every Activity run. In addition, Processing Sequence run history can be purged using the Database Housekeeping function. The run history for Processing Sequence runs that have been purged is no longer available.

# 2.5.3 Transformation Map Details

The details of a Transformation Map consist of basic identification and status information:

📄 Details 🛛 🔍 Data intercha	ange 🛛 😫 Parameters 🛛 🔂 Run history 🛛 🔾 Where used 🛛 🛄 No	otes 🛛 🐼 Audit
ID	DEM_ORDER_DB2	Save
Description	Order XML to IBM i DB2	Refresh
Мар Туре	Demonstration maps	<b>N</b>
Restartable	Yes	▶ R <u>u</u> n
Status	Active	

Basic identification and status information for a Transformation Map definition is as follows:

#### ID

An identifier to uniquely identify this Transformation Map.

#### Description

This should describe the purpose or use of the Transformation Map.

#### Мар Туре

You can assign a map type to the Transformation Map using the dropdown list .

**Note:** Transformation Map types are used to group and categorize transformation maps that have a similar function or purpose, particularly for when transformation maps are linked to trading partner definitions. Map types are user-defined. The dropdown list will contain Transformation Map types only if you have defined types in LANSA Composer. Refer to Code Maintenance for information on defining map types.

#### Restartable

Yes or No. This value determines whether a Processing Sequence run that fails while executing this Transformation Map can be restarted (if it is otherwise eligible).

**Yes** indicates the Transformation Map can be restarted.

**No** indicates that this Transformation Map cannot be restarted. A Processing Sequence that fails on this Transformation Map will not be eligible to be restarted.

#### Status

Active or Inactive. Transformation Maps cannot be used in a processing sequence when they are Inactive.

## 2.5.4 Transformation Map Data Interchange Attributes

LANSA Composer defines and stores a set of data interchange attributes for each Transformation Map. These are defined to accommodate some of the attributes commonly used in EDI and other forms of data interchange.

While LANSA Composer pre-defines fields to hold these attributes, the program does not enforce the entry of the fields, nor does it validate values entered. To permit maximum implementation flexibility, LANSA Composer leaves the use and values of these attributes to you according to your own circumstances and conventions.

Some of these values are, however, important for the operation of the transaction document framework as supplied. Refer to Transformation Map Data Interchange Attributes for more information.

To review or change the data interchange attributes for a Transfomation Map, select the required Transfomation Map in the Transfomation Maps list and then click the *Data interchange* tab in the command handler area.

📄 Details 🤷 Data interchange 💉	🕏 Parameters   🔁 Run history   🔍 Where used   💟 Notes   🧭	Audit
Map direction	Inbound	<u>S</u> ave
Map standard	EDI X12 004010 💌	efresh
Map transaction type	850	
Import processor	EDI850IMP	
Export processor		

To modify a value, type the new value. Remember to click *Save* when complete to save your changes.

For information about accessing data interchange attribute values for a Transformation Map in a processing sequence, refer to Transformation Map (\*TRANSFORM) Built-in Variable Qualifiers.

### 2.5.5 Edit And Prepare Transformation Map

Beyond entering details in the LANSA Composer, there are two steps necessary to define a Transformation Map before it can be used in a processing sequence:

- 1. Press *Edit* at the Transformation Map *Details* to proceed to edit the Transformation Map in the mapping tool to perform the mapping required for your application. Refer to Edit Transformation Map for further information.
- 2. Press *Prepare* at the Transformation Map *Details* after you have finished editing your map and its definition has been saved. This step will generate and build the executable implementation of the map that will be included in a Processing Sequence. Refer to Prepare Transformation Map for further information.

When you have successfully completed these steps, the *Parameters* tab will show the new parameter information for the map, and you can define and run Processing Sequences that use the Transformation Map.

# **Edit Transformation Map**

When you *Edit* your Transformation Maps, LANSA Composer will start MapForce to provide the visual mapping tools. Following is brief overview of the essential steps to define a typical Transformation Map using the mapping tool. To make effective use of this tool, refer to Learn about the Mapping Tool for the tutorials and documentation that may help you.

# **Supported Functionality Reminder**

Before MapForce starts you may see the window shown below. This window reminds you that not all MapForce functions are supported for use with LANSA Composer.

Edit Transformation Map           LANSA Composer supports a subset of the full range of functionali available in the MapForce mapping tool.		ge of functionality	×
	Click Help for more information.	Help	J
	Click Continue to proceed to edit your transformation map.	Continue	
	Click Cancel if you don't want to edit your transformation map now.	Cancel	
Don't show th	nis again		

For more information about what functions are or are not supported for use with LANSA Composer, refer to Supported Functionality of the Mapping Tool.

The first time you edit a Transformation Map you will see this window. However, you are given the option to bypass this window. You may choose to hide it for subsequent use by selecting the *Don't show this again* option.

# Specifying Source and Target Components in the Mapping Tool

When you add source and target components to your map in the mapping tool, it is a good idea to identify the locations of the components or the connection details in a way that applies universally in your organization. For example, you should identify the location of components from the file system using a network path or a mapped drive that is accessible to all clients. When adding database components use standard drivers, DSN and other details that will be available on all your client computers.

Following these suggestions will ensure that if someone else needs to open the map to review or change it, they will be able to do so without error and avoid having to re-locate the components used in the map.

## **Prepare Transformation Map**

After you have created or changed the definition of your Transformation Map using the mapping tool, you need to prepare it for use before it can be executed in a processing sequence.

To do this, simply click the *Prepare* button at the Transformation Map *Details*. and then click *Prepare* in the *Prepare Transformation Map* window. Preparing your map may take a few minutes.

repare	Transformation Map		>
X	Transformation map Description	ITUT_ORDERS	
	Preparing transformation m	ар	Prepare
			Cancel
✓ Check ✓ Preprod	folders and files cess transformation map defin	nition	
Gener	ate code for transforma	ntion map	
Cotolog	e code for transformation map ave information for transformation	P Nion man	
Deploy	transformation map executal	ble files to LANSA Composer server	

You may briefly see the MapForce window during the Prepare, as LANSA Composer loads and processes your Transformation Map to generate code that will execute the mapping. This is normal and you should wait for the MapForce window to close again and the Prepare to continue.

When the *Compile code for transformation map* step completes successfully you will see a window like this:



You should click OK to allow the remaining steps to proceed.

When preparation is complete, you may click the *Show Log* button to display a detailed log of the steps completed. The log will be useful for diagnosing

failures in the preparation..

# What Does Prepare Do?

The main steps are:

- Generates Java code that implements the Transformation Map as defined in the mapping tool. (You may briefly see the MapForce window appear during this step).
- Compiles the generated Java code.
- Determines the parameters that the Transformation Map requires and catalogues information about the map.

When it is complete, LANSA Composer places the resulting .jar (Java archive) file in the /composer/<*partition*>/map directory of the LANSA Integrator instance. This is the executable implementation of the Transformation Map.

**Note:** When your transformation map uses database components that connect to IBM DB2 for i5/OS database tables, LANSA Composer will remove the library qualifier from the generated Java code when you prepare the transformation map. You must setup your database configuration so that the required files can be found at run-time. For more information, refer to Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS .

When you have successfully prepared your Transformation Map, the new or revised parameters for the Transformation Map can be reviewed on the *Parameters* tab. The actual parameters shown for the Transformation Map depend upon the map definition in the mapping tool. The catalogued information for the map can be viewed on the *Catalogued* tab.

# **3. Processing Sequences**

Processing sequences allow you to combine Activities and Transformations with Processing Directives such as Loops and conditions to complete a business process. Combining these elements with Variable Transport, Database Configurations and Trading Partner information permits great flexibility in accomplishing complex multi-step processing without programming.

3.1 Anatomy of a Processing Sequence
3.2 Work With Processing Sequences
3.4 Use the Processing Sequence Editor
3.5 Run a Processing Sequence
Maintain Attachments
Define or Review Notes
Review Audit Details

# 3.1 Anatomy of a Processing Sequence

The following screen picture shows some of the major concepts supporting a Processing Sequence. The numbers refer to the following paragraphs that elaborate on the highlighted concepts:



# **1.** Activities and Transformation Maps provide the business functionality

The main purpose of Processing Sequences in LANSA Composer is to run one or more Activities and Transformation Maps in a defined order. The simplest Processing Sequences might simply perform a single Activity or Transformation.

		*** Beginning of processing sequence ***
÷	Activity	
	Name:	FTP_INBOUND
	Description:	FTP Get files from remote host
·		*** End of processing sequence ***

More typically a Processing Sequence will combine one or more Activities and/or Transformation Maps in order to complete a business process. Refer to 3.1.1 Activities and Transformation Maps for information.

# 2. Variables provide Communication between Activities

In any non-trivial Processing Sequence, it will probably be necessary to communicate information between the Activities and Transformation Maps that make up the Processing Sequences. LANSA Composer provides the Processing Sequence variable pool for this purpose. Refer to 3.1.2 Variables and 3.1.3 Built-in Variables for details.

# 3. Processing Sequence Directives provide Powerful Orchestration

In order to perform complex multi-step business processes, it is not enough to simply string a sequence of Activities or Transformations together. You need some means of controlling their execution with loops, conditioning and other constructs.

		*** Beginning of processing sequence ***
÷	Activity	
	Name:	FTP_INBOUND
	Description:	FTP Get files from remote host
÷	Loop	
	List:	FILEPATH
	Loop variable:	THISFILE
ė.	🍙 Transform	
	Name:	DEM_ORDER_DB2
	Description:	order.xml to DB2
		*** End of processing sequence ***

LANSA Composer provides a set of Processing Sequence Directives for this purpose. Refer to 3.1.4 Processing Directives for more information.

# 4. Processing sequence parameters for Run-Time Flexibility

Processing sequences can receive parameters that become variables in the variable pool. The use of parameters enhances the flexibility of Processing Sequences by permitting the use of a single Processing Sequence with different inputs to satisfy multiple variations of a business process case. Refer to 3.1.5 Parameters for more information.

#### Also see

- 3.1.1 Activities and Transformation Maps
- 3.1.2 Variables
- 3.1.3 Built-in Variables
- 3.1.4 Processing Directives
- 3.1.5 Parameters
- 3.2 Work With Processing Sequences

## **3.1.1 Activities and Transformation Maps**

You can use any shipped or custom Activities and Transformation Maps in a Processing Sequence. The simplest Processing Sequences might simply perform a single Activity or Transformation.

		*** Beginning of processing sequence ***
Ė	Activity	
	Name:	FTP_INBOUND
	Description:	FTP Get files from remote host
		*** End of processing sequence ***

When you add an Activity or Transformation Map to a Processing Sequence, the detailer window in the *Processing sequence editor* shows the parameters that are defined for the Activity or Transformation.

Del	tails	Parameter	s			
Augusta			Variable or value for parameter	Parameter name	Parameter description	Parameter class
1	*	Inbound	'DEM_FTP_IN'	 FTPCONFIG	FTP Configuration ID	FTP inbound configu
2		Inbound		REMOTEDIRECTORY	Override Remote Directory to GET file fror	General
3		Inbound		LOCALDIRECTORY	Override Local Directory into which file pla	Folder path
4		Inbound		GETLIKE	Override 'GET selection' for filename mask	General
5		Outbound		FILELIST	List of file retrieved - file name	General
6		Outbound		 FILEPATH	List of file retrieved - full path & name	File path

In this window you can specify values that are to be used for the parameters in two ways:

1. You can enter a numeric or alphanumeric (according to the use of the parameter) literal value. If you enter an alphanumeric literal value you can surround it with quote marks. The quote marks do not become part of the parameter value that the Activity receives. If you need to embed quote marks in the parameter value, you should surround the value with quote marks and double-up the embedded quote marks.

If the parameter class is other than General, you can press F4 or click the adjacent Prompt (...) button to browse and choose from a list of possible values. For example, if the parameter class is *Trading Partner*, LANSA Composer will display a list of defined Trading Partners. Or if the parameter class is *File path*, LANSA Composer will allow you to browse and choose from files present in the server file system.

2. You can specify the name of a variable or built-in variable that supplies or receives the value for the parameter. To specify a variable, precede the variable name with an ampersand (&) – for example &VARIABLE1. To specify a built-in variable, enter the name, including the asterisk, as shown

on the *Built-ins* tab – do not precede a built-in variable name with ampersand.

If the variable you wish to use is already known to the Processing Sequence, you can simply drag it from the *Variables* tab.

Each parameter for an Activity (or the variable specified for it) becomes part of a pool of variables that are available to all Activities, Transformation Maps and Processing Directives in the Processing Sequence. In this way any parameters that are output from one Activity might be used as input to another Activity or Transformation Map later in the same Processing Sequence.

# 3.1.2 Variables

The Processing Sequence contains a pool of variables that may be used as parameters to Activities and Transformation Maps and may be tested and manipulated using Processing Sequence Directives.

Variables may be displayed and accessed using the *Variables* tab in the *Processing sequence editor*.



Some important points to note about Processing Sequence variables:

- Variables implicitly become available in the variable pool when they are used or referenced as parameters or in Processing Sequence Directives. There is no explicit definition of variables in the Processing Sequence.
- Variable names are not case-sensitive for example &ftpconfig and &FTPCONFIG reference the same variable.
- Variables are loosely-typed it is the responsibility of the Activity and Processing Sequence designers to ensure that values in variables are appropriate for their intended use.
- Variables may be used as "lists" simply by indexing references to them. Refer to Lists for further information.

## Variable Usage

Variables can be used to maintain state and to pass information from one Activity to a later Activity in the Processing Sequence. For example, the FTP\_INBOUND shipped Activity returns a list of files retrieved by the FTP operation. This list may be used to control later Directives and Activities in the Processing Sequence.

Mostly, processing sequence variables are used to hold variable data that is used

to orchestrate the process - for example, paths to transaction documents that are being processed.

It is less common that the processing sequence variables are used to hold application data items, such as the current order or customer number. Most typically, this type of data is and should be processed by the Activities and Transformation maps used in the Processing Sequence and/or LANSA functions and/or Java or 3GL programs called by the Processing Sequence.

Sometimes, however, it is desirable or necessary to fill a processing sequence variable or variable lists with such application data items. For example, after transforming a transaction document containing an in-coming sales order, it may be necessary that the Processing Sequence "know" an identifying key such as a batch or order number or the customer number, in order to pass it as a parameter to, say, a LANSA function that will further process the order.

There are several techniques available to accomplish this. Some of the most common include:

- using a transformation map, extract the required data from the transaction document file or from a database to a CSV (comma-separated variables) file
- use the FOR\_EACH\_TXTLIN or FOR\_EACH\_CSVROW activities to read the contents of a text or CSV file
- write a function, program or custom activity to extract and return the required information
- on IBM i servers, you can use data areas and message queues to exchange information with other programs or processes using supplied activities such as GET\_DTAARA and PUT\_DTAARA or MSGQ\_SEND and MSGQ\_RECEIVE.

In addition, LANSA Composer defines an XML file format and provides activities for saving and loading processing sequence variables and for transforming application data to or from that file format. For more information, refer to:

Save, Load and Transform Processing Sequence Variables

## Lists

References to variables in the variable pool may be indexed. In this way, variable lists may be created. The lists may, in turn, be processed iteratively using a Loop Processing Directive.

For example, the FTP\_INBOUND shipped Activity defines an output parameter FILEPATH which contains a list of the paths and file names of all files retrieved in the FTP operation. The Activity Processor is written to make indexed references to the &FILEPATH variable in order to build up the list. The entire list of FILEPATHs is then available to subsequent Activities and Processing Directives using indexed references to the &FILEPATH variable.

The Processing Sequence example shown below uses the FTP\_INBOUND Activity and then executes a Transformation Map in a loop for each file received.

		*** Beginning of processing sequence ***
÷	Activity	
	Name:	FTP_INBOUND
	Description:	FTP Get files from remote host
÷	Loop	
	List:	FILEPATH
	Loop variable:	THISFILE
Ė.	💩 Transform	
	Name:	DEM_ORDER_DB2
	Description:	order.xml to DB2
·		*** End of processing sequence ***

Any variable may be used as a list simply by indexing references to it. However, Activity parameters may be explicitly flagged as being lists and will be represented by a different icon in the *Variables* tab in the *Processing sequence editor*.

Indexed references to variables should use contiguous indices starting from 1. The Loop Directive and the supplied Activities will only process instances that meet this condition.

#### Save, Load and Transform Processing Sequence Variables

LANSA Composer defines a document format and provides activities for saving and loading processing sequence variables and for transforming application data to or from that format. This support provides a further means to populate and exchange data in processing sequence variables. It facilitates solutions to scenarios such as:

• The Processing Sequence needs access to certain variable data from a transaction document file

Using a Transformation Map, the transaction data can be mapped to the format of LANSA Composer's Process Sequence Variables (PSV) Files. Then using the LOAD\_PSVSET activity, the Processing Sequence can load the required variables from the PSV file.

• Certain variable data originating in the Processing Sequence needs to be written to a transaction document file

The Processing Sequence may save the variable data using the SAVE\_PSVSET activity and then execute a Transformation Map that transforms the variable data from the format of LANSA Composer's Process Sequence Variables (PSV) Files to the transaction document file or database.

• One Processing Sequence performs Activities that populate a set of Processing Sequence variables and/or variable lists that are then used in a second Processing Sequence

The first Processing Sequence may save the variable data using the SAVE\_PSVSET activity and then start or submit the second Processing Sequence which will load the variable data using the LOAD\_PSVSET activity.

The following headings provide further information about saving, loading and transforming Processing Sequence variables:

Process Sequence Variables (PSV) Files Save and Load a PSV File Transform To or From a PSV File

#### **Process Sequence Variables (PSV) Files**

LANSA Composer defines a document format to hold or receive Processing Sequence variables and variable lists. Documents that conform to LANSA Composer's format may be used with the LOAD\_PSVSET and SAVE\_PSVSET activities and in Transformation Maps.

The document format is actually XML and it is described in a Document Type Definition (DTD) that can be found at

http://www.lansa.com/schemas/lc3psv.dtd. A copy of the DTD, along with a sample PSV file is also installed with LANSA Composer.

This DTD defines LANSA Composer's Processing Sequence Variables (PSV) file format. The 'PSV' file extension is only a suggestion, not a requirement.

The following is a sample of a PSV file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE psVariables SYSTEM
```

```
"http://www.lansa.com/schemas/lc3psv.dtd">
```

<ps Variables>

<psvSet id="1" comment="List containing the names of the colours of the
rainbow in English">

```
<psVariable name="rainbow" index="1" value="Violet"/>
<psVariable name="rainbow" index="2" value="Indigo"/>
<psVariable name="rainbow" index="3" value="Blue"/>
<psVariable name="rainbow" index="4" value="Green"/>
<psVariable name="rainbow" index="5" value="Yellow"/>
<psVariable name="rainbow" index="6" value="Orange"/>
<psVariable name="rainbow" index="7" value="Red"/>
```

</psvSet>

<psvSet id="2" comment="List containing the names of the colours of the
rainbow in French">

```
<psVariable name="rainbow" index="1" value="Violet"/>
<psVariable name="rainbow" index="2" value="Indigo"/>
<psVariable name="rainbow" index="3" value="Bleu"/>
<psVariable name="rainbow" index="4" value="Vert"/>
<psVariable name="rainbow" index="5" value="Jaune"/>
<psVariable name="rainbow" index="6" value="Orange"/>
<psVariable name="rainbow" index="7" value="Rouge"/>
</psvSet>
```

</psVariables>

Some points to note about the PSV file format:

- Like any XML file, the root element <psVariables> can occur only once in the file.
- The <psVariables> root element must contain at least one or more Processing Sequence Variable "sets" defined by a <psvSet> element.

The possibility for a PSV file to contain more than one PSV Set is important in connection with the LOAD\_PSVSET activity, which has iterator capabilities.

• Within each <psvSet> element there may be one or more <psVariable> elements, each of which defines a Processing Sequence variable, its value and an optional index for use with variable lists.

A Processing Sequence Variables (PSV) file does not contain built-in variables or their values.

#### Save and Load a PSV File

LANSA Composer provides the following activities specifically for saving and loading Processing Sequence Variables in the format of LANSA Composer's Process Sequence Variables (PSV) Files.

Activity ID	Description
LOAD_PSVSET	Load processing sequence variables from a PSV file.
SAVE_PSVSET	Save processing sequence variables to a PSV file.

Refer to the activity descriptions for further details.
## Transform To or From a PSV File

Because LANSA Composer's Process Sequence Variables (PSV) Files are XML files, you can use them as the source or target of a Transformation Map.

To use a PSV file in a Transformation Map:

- 1. Select *XML Schema/File* from the *Insert* menu in the mapping tool;
- 2. In the *Open* dialogue, navigate to and select the supplied *lc3psv.dtd* file or the *lc3sample.psv* file, or type the URL of the DTD file: *http://www.lansa.com/schemas/lc3psv.dtd*;
- 3. You can then map to or from the PSV file component in the usual way.



Note: When using the PSV file component as the target of a mapping and you wish to map more than one variable, you will usually need to use the *Duplicate Input* command in the mapping tool to create multiple instances of the *psVariable* element node that you can map to. In the example above you can see the node () **psVariable (2)** was defined this way in order to map the second variable ("TUT\_CUSNUM").

### 3.1.3 Built-in Variables

LANSA Composer provides a set of special built-in variables that can be referenced in any Processing Sequence. Built-in variables may be displayed and accessed using the *Built-ins* tab in the *Processing sequence editor*.



The built-in variables that LANSA Composer provides are listed below. Those built-in variables that have *Yes* in the Writeable column (below) may be used as the receiving variable in an Assign Directive. All other built-in variables are read-only.

Name	Description
*ACTIVITY_RC	The return code from the last executed Activity.
*ACTIVITY_SEV	The severity corresponding to the return code from the last executed Activity.
*JOBMODE	Current job mode (B=batch, I=inter) (Derived directly from the LANSA system variable of the same name)
*JOBNAME	Current job name

	(Derived directly from the LANSA system variable of the same name)
*JOBNUMBER	Current job number
	(Derived directly from the LANSA *JOBNBR system variable)
*JOBUSER	Current job user name
	(On an IBM i server, this may be different to the *USER built-in variable value if the current user for the job changed since the job was started.)
*LANGUAGE	Current LANSA language code
	(Derived directly from the LANSA system variable of the same name)
*LANSADTALIB	LANSA system data/file library
	(Derived directly from the LANSA system variable of the same name)
*LANSAPGMLIB	LANSA system program library
	(Derived directly from the LANSA system variable of the same name)
*LASTERROR.ID	The message id of the error message following the last error.
	(This is intended for use in Catch blocks to support further interrogation of the error condition. The "last" error message is defined as the FIRST error message logged after an activity sets the error (ER) result status,)
*LASTERROR.TX1	The first level message text of the error message following the last error.
	(This is intended for use in Catch blocks to support further interrogation of the error condition. The "last" error message is defined as the FIRST error message logged after an activity sets the error (ER) result status,)

*LASTERROR.TX2	The second level message text of the error message following the last error.
	(This is intended for use in Catch blocks to support further interrogation of the error condition. The "last" error message is defined as the FIRST error message logged after an activity sets the error (ER) result status,)
*NOW_UTC *NOW_LOCAL	The current UTC and local date and time formatted in ISO format with up to three decimal positions: YYYY-MM-DD hh:mm:ss.nnn
*PARTITION	Current LANSA partition
	(Derived directly from the LANSA system variable of the same name)
*PARTDTALIB	Current LANSA partition's data/file library (Derived directly from the LANSA system variable of the same name)
*PARTPGMLIB	Current LANSA partition's program library (Derived directly from the LANSA system variable of the same name)
*PROCESS_ID	The identifier or name of the running Processing Sequence
*PROCESS_II	The unique internal identifier of the running Processing Sequence. (The internal identifier is a 32 character value uniquely assigned by LANSA Composer. You can see the unique identifier for a Processing Sequence on the <i>Audit</i> tab.)
*PROCESS_LOGLEVEL	The logging level in effect for the processing sequence run. Initially, the logging level is taken from the System Settings but may vary during the run, either automatically following an error or by explicit assignment of a new

	logging level. This variable can contain or be set to values corresponding to the logging levels that can be selected as the default in System Settings: *AUTO (Automatic), *MIN (Minimum), *NORMAL (Normal) or *MAX (Maximum).
*PROCESS_JSMTRACE	Specifies (Y/N) whether LANSA Integrator tracing is in effect for the processing sequence run. Initially, the value is taken from the System Settings but may vary during the run by explicit assignment of a new value to this built- in variable. Note that changes to the value will only affect LANSA Integrator sessions begun after the change is effected. Many activities processors remain active and possibly keep a JSM session open for subsequent use of the same activity. To be sure of affecting such cases, the change to the built-in value variable needs to be made before the first use of the affected activity
*PROCESS_NEST	The nested level of the running processing sequence. Usually the value of this is 1. When a processing sequence is run from another processing sequence using the Processing Sequence directive, the value reflects the level of such nesting.
*PROCESS_RC	The return code for the Processing Sequence.
*PROCESS_RUN	The run number assigned to the current Processing Sequence run.
*PROCESS_RSM_UTC *PROCESS_RSM_LOCAL	The UTC and local date and time of the last restart for the current Processing Sequence run formatted in ISO format with up to three decimal positions: YYYY-MM-DD hh:mm:ss.nnn
*PROCESS_SEV	The severity corresponding to the return code

for the Processing Sequence.

*PROCESS_STR_UTC *PROCESS_STR_LOCAL	The UTC and local start date and time for the current Processing Sequence run formatted in ISO format with up to three decimal positions: YYYY-MM-DD hh:mm:ss.nnn
*SHUTDOWN	The *SHUTDOWN built-in variable yields a value of 'Y' when one of the following is true: 1) the system, or the subsystem or the job in which the processing sequence is executing is subject to a controlled end request - for example, by PWRDWNSYS, ENDSBS or ENDJOB commands or equivalent with OPTION(*CNTRLD) NB: This applies on IBM i servers only. There is no equivalent support on Windows servers for this type of "shutdown" request. 2) A LANSA Composer user has requested "controlled end" for an "active" processing sequence run, for example, by using the "End"
	command or toolbutton in the Processing Sequence Log window The processing sequence controller does NOT automatically respect such requests. It is up to the solution designer to build in such support where appropriate by reference to this new *SHUTDOWN built-in variable. It may not always be necessary, but it is advised for processing sequences that are intended to be long-running processes. This would usually include, for example, any processing sequence that uses the new WATCH_MSGQ, WATCH_DIRECTORY or WATCH_DTAQ activities, or one that implements similar "monitor" style processing using LOOPs or other constructs.

\*SYS\_DEFAULT\_DIR

The path specified in the LANSA Composer

	system settings for the default trading partner linked directory.
*SYSTEMTYPE	The server system type (AS400, WINNT or UNIX)
	(Derived directly from the LANSA *CPUTYPE system variable)
*TRADINGPARTNER	The current trading partner identifier. This built-in variable may be extended with a range of qualifiers to access attributes and linked directories, Configurations and Transformation Maps for the trading partner.
	Note that while the *TRADINGPARTNER built-in variable is writeable, the qualified forms that provide access to the attributes of the current trading partner are not writeable. Refer to Using the Trading Partner Built-in Variables for more information.
*TRADINGPARTNERS	The list of active trading partner identifiers. This built-in variable may optionally be extended with trading partner group qualifiers to access subsets of the trading partners list. For example, *TRADINGPARTNERS.GROUP.CUSTOMER would provide the list of active trading partners that belong to the CUSTOMER trading partner group. This built-in variable (or its qualified forms) may be used as the list name for a Loop Directive to create a loop that performs processing for all defined trading partners (or subsets of trading partners). Refer to Using the Trading Partner Built-in Variables for more information
*TRANSFORM	The current transformation map identifier. This built-in variable may be extended with a range

	of qualifiers to access attributes of the transformation map. Note that while the *TRANSFORM built-in variable is writeable, the qualified forms that provide access to the attributes of the current transformation map are not writeable. Refer to Using the Transformation Map Built- in Variables for more information.
*TXDOC	The current transaction document identifier (number). This built-in variable may be extended with a range of qualifiers to access attributes of the transaction document envelope.
	Note that while the *TXDOC built-in variable is writeable, the qualified forms that provide access to the attributes of the current transaction document envelope are not writeable.
	Refer to Using the Transaction Document Built-in Variables for more information.
*TZ_ABBR	On IBM i servers, this built-in variables provides the short name of the time zone used by the job – for example, AEST. On Windows servers, the description for the current time zone is provided (the same value as *TZ_DESC and *TZNAME).
*TZ_DESC	On IBM i servers, this built-in variables provides the time zone description used to calculate job time – for example, QP1000AES3. On Windows servers, the description for the current time zone is provided (the same value as *TZ_ABBR and *TZNAME).
*TZ_NAME	On IBM i servers, this built-in variables provides the long name of the time zone used

	by the job – for example, Australian Eastern Standard Time. On Windows servers, the description for the current time zone is provided (the same value as *TZ_ABBR and *TZDESC).
*TZ_OFFSET	The current offset from Coordinated Universal Time (UTC), formatted as a string in the form +HH:MM – for example, +10:00.
*TZ_OFFSETMM	The current offset from Coordinated Universal Time (UTC), in minutes – for example, 600.
*USER	Current user ID
	(Derived directly from the LANSA system variable of the same name)

#### Also see

Using the System Property Built-in Variables Using the Trading Partner Built-in Variables Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers Using the Transformation Map Built-in Variables Transformation Map (\*TRANSFORM) Built-in Variable Qualifiers Using the Transaction Document Built-in Variables Transaction Document (\*TXDOC) Built-in Variable Qualifiers

### **Using the System Property Built-in Variables**

Values of installation-defined system properties may be accessed (or assigned if defined as *writeable*) by using the following built-in variable form:

```
*SYSTEM.PROPERTY.<property name>
```

where <property name> is the name of the installation-defined system property. Refer to System Properties for information on defining system properties.

In the following example, the *\*system.property.myproperty* form is used in the details for an activity to access the value of the installation-defined system property named *myproperty*:

_	Det	ails	Parameters	3		
				Variable or value for parameter	Parameter N	Parameter description
	1	×	Inbound	*tradingpartner.property.myproperty	VARIABLE	Specify the (variable) value to
	<					>

**Note:** The use and values of these properties is subject to your own circumstances and conventions. LANSA Composer does not enforce the entry of the fields corresponding to these properties, nor does it validate any values entered.

### **Using the Trading Partner Built-in Variables**

The trading partner information may be referenced at run-time (in a Processing Sequence) in several ways:

- When processing is being performed for a particular trading partner, the linked directories, Transformation Maps and/or Configurations can be accessed through qualified forms of the \*TRADINGPARTNER built-in variable and used as variable inputs to the Activities and Transformation Maps to be executed. Refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers for more information.
- A Processing Sequence may contain a loop based on the built-in \*TRADINGPARTNERS variable (or one of the \*TRADINGPARTNERS.GROUP.xxx variables) to iteratively perform the same processing for all defined trading partners (or trading partners that belong to the specified trading partner group).



• If you wish to process a single trading partner in your Processing Sequence but still require access to the trading partner information using the qualified forms of the \*TRADINGPARTNER variable then you must first assign a value to the \*TRADINGPARTNER built-in variable. The value you assign must be the identifier of a trading partner defined in the LANSA Composer system.



In order to successfully use the \*TRADINGPARTNER built-in variable and its qualified forms, a valid value must have been assigned to it in one of the above ways.

# Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers

The \*TRADINGPARTNER built-in variable may be extended with a range of qualifiers to access the attributes and properties of the trading partner and the directories, Transformation Maps and/or Configurations linked to the trading partner. The available qualified forms of the \*TRADINGPARTNER built-in variable may be displayed and accessed by expanding the corresponding item listed on the *Built-ins* tab in the *Processing sequence editor*.

**Note:** While the \*TRADINGPARTNER built-in variable is writeable, the qualified forms that provide access to the attributes of the current trading partner are not writeable.

### **Trading Partner Attributes**

The base attributes for the current trading partner may be accessed using the following qualifiers with the \*TRADINGPARTNER variable:

Qualifier	Description	
.II	The unique internal identifier of the trading partner. (The internal identifier is a 32 character value uniquely assigned by LANSA Composer. You can see the unique identifier for a trading partner on the <i>Audit</i> tab.)	
.ID	The identifier or name of the trading partner definition This is the value returned for the *TRADINGPARTNER built-in variable if no qualifier is specified. In other words * TRADINGPARTNER yields the same value as *TRADINGPARTNER.ID.	
.STATUS	The status code (A=active, I=inactive) of the current trading partner. <b>Note:</b> When processing Trading Partners in a loop using the *TRADINGPARTNERS list variable,	

	only active Trading Partners are processed. However, if you explicitly set the trading partner using the *TRADINGPARTNER variable, you can use this attribute to check the status of the Trading Partner.	
.NM	The description of the trading partner definition.	
.A1 .A2 .A3	The street address lines 1, 2 and 3 for the trading partner.	
.PH	The telephone number for the trading partner.	
.FX	The fax number for the trading partner.	
.EM .EM2 .EM3	The email addresses for the trading partner.	
.TXDOC_RECEIVE	The identifier (name) of a processing sequence associated with this trading partner to send outbound transaction documents, if specified.	
.TXDOC_SEND	The identifier (name) of a processing sequence associated with this trading partner to receive inbound transaction documents, if specified.	

In the following example, the *\*tradingpartner.id* form is used in an If condition to condition a part of the Processing Sequence:



### **Trading Partner Properties**

Values of installation-defined properties for a Trading Partner may be accessed by using the following qualified forms of the \*TRADINGPARTNER built-in variable:

\*TRADINGPARTNER.PROPERTY.<property name>

where <property name> is the name of the installation-defined Trading

Partner property. Refer to Code maintenance for information on defining Trading Partner properties.

In the following example, the *\*tradingpartner.property.myproperty* form is used in the details for an activity to access the value of the installation-defined property named *myproperty* for the current Trading Partner:



Note: The use and values of these properties is subject to your own circumstances and conventions. LANSA Composer does not enforce the entry of the fields corresponding to these properties, nor does it validate any values entered.

### **Trading Partner Data Interchange Attributes**

The data interchange attributes for the current trading partner may be accessed using the following qualifiers with the \*TRADINGPARTNER variable:

Qualifier	Description
.EDI.CA	Communication agreement
.EDI.IAP	Inbound archive file prefix
.EDI.IRA	Inbound receiver routing address
.EDI.IRI	Inbound receiver ID
.EDI.IRQ	Inbound receiver qualifier
.EDI.ISA	Inbound sender routing address
.EDI.ISI	Inbound sender ID
.EDI.ISQ	Inbound sender qualifier
.EDI.OAP	Outbound archive file prefix
.EDI.OAR	Outbound application receiver

.EDI.OAS	Outbound application sender
.EDI.ORA	Outbound receiver routing address
.EDI.ORI	Outbound receiver ID
.EDI.ORQ	Outbound receiver qualifier
.EDI.OSA	Outbound sender routing address
.EDI.OSI	Outbound sender ID
.EDI.OSQ	Outbound sender qualifier
.EDI.RPQ	Recipient's password qualifier
.EDI.RPW	Recipient's password
.EDI.SPLITIGNORECR	EDI split ignore carriage returns
.EDI.SPLITMAX	EDI split maximum transactions
.EDI.XTP	Exchange trading partner

Note: The use and values of these attributes is subject to your own circumstances and conventions. LANSA Composer does not enforce the entry of the fields corresponding to these attributes, nor does it validate any values entered.

### **Trading Partner Linked Directories**

Directories linked to a trading partner may be accessed using the following qualified form of the \*TRADINGPARTNER built-in variable:

### \*TRADINGPARTNER.DIR.<directory type>

where <directory type> is a directory type code defined for the system. Refer to Code maintenance for information on defining directory types.

In the following example, the \*tradingpartner.dir.iftp form is used in the details for an FTP\_INBOUND Activity to access the inbound FTP directory for the current trading partner:

Det	ails	Parameters			
			Variable or value for parameter	Parameter N	Parameter description
1	*	Inbound	*tradingpartner.ftpin	FTPCONFIG	FTP Configuration ID
2		Inbound	*tradingpartner.dir.iftp	REMOTEDIR	Override Remote Directory to
3		Inbound		LOCALDIREC	Override Local Directory into
4		Outbound		FILELIST	List of file retrieved - file name
5		Outbound		FILEPATH	List of file retrieved - full path
<					

## **Trading Partner Linked Transformation Maps**

Transformation maps linked to a trading partner may be accessed using the following qualified form of the \*TRADINGPARTNER built-in variable:

### \*TRADINGPARTNER.MAP.<map type>

where <map type> is a map type code defined for the system. Refer to Code Maintenance for information on defining map types.

In the following example, the \*tradingpartner.map.orderin form is used in the details for a TRANSFORM Activity to access the Transformation Map of type orderin linked to the current trading partner:

D	eta	ails	Parameters	8			
				Variable or value for parameter	Parameter Name	Parameter description	^
	1	×	Inbound	*tradingpartner.map.orderin	TRANSFORMID	ID of the transformation map to run	
	2		Inbound	&THISFILE	TRANSFORMP1	Parameter 1 for transformation map	
	3		Inbound	'EXAMPLEDB_400'	TRANSFORMP2	Parameter 2 for transformation map	
	4	.0%	Inbound		TRANSFORMP3	Parameter 3 for transformation map	
	5		Inbound		TRANSFORMP4	Parameter 4 for transformation map	~
<						) >	

#### Notes:

1. To execute a Transformation Map linked to a trading partner you must use the supplied TRANSFORM activity. Refer to TRANSFORM for information on this activity.

2. More than one Transformation Map of the same type may be linked to a Trading Partner. This form of the \*TRADINGPARTNER built-in variable yields the first Transformation Map of the specified type. A more flexible and powerful means of identifying a Transformation Map linked to a Trading Partner is provided by the supplied FIND\_TPMAP activity. Refer to FIND\_TPMAP for information on this activity.

### **Trading Partner Linked Configurations**

Configurations linked to a trading partner may be accessed using the following qualified forms of the \*TRADINGPARTNER built-in variable:

Qualifier	Description
.DB	The database Configuration linked to the trading partner definition.
.FTPIN	The FTP inbound Configuration linked to the trading partner definition.
.FTPOUT	The FTP outbound Configuration linked to the trading partner definition.
.FTPCMDS	The FTP command list Configuration linked to the trading partner definition.
.HTTPIN	The HTTP inbound Configuration linked to the trading partner definition.
.HTTPOUT	The HTTP outbound Configuration linked to the trading partner definition.
.MSG	The Messaging Configuration linked to the trading partner definition.
.LANSASYS	The LANSA system Configuration linked to the trading partner definition.
.POP3	The POP3 mail Configuration linked to the trading partner definition.
.SMS	The SMS configuration linked to the trading partner definition.
.SMTPMAIL	The SMTP mail Configuration linked to the trading partner definition.
.SMTPSERVER	The SMTP server Configuration linked to the trading partner definition.

In the following example, the \*tradingpartner.ftpin form is used in the details for an FTP\_INBOUND Activity to access the inbound FTP directory for the current trading partner:

De	tails	Parameters			
			Variable or value for parameter	Parameter N	Parameter description
1	-	Inbound	*tradingpartner.ftpin	FTPCONFIG	FTP Configuration ID
2		Inbound	*tradingpartner.dir.iftp	REMOTEDIR	Override Remote Directory to
3		Inbound		LOCALDIREC	Override Local Directory into
4		Outbound		FILELIST	List of file retrieved - file name
5		Outbound		FILEPATH	List of file retrieved - full path
<			Ш		

### Using the Transformation Map Built-in Variables

Transformation map information may be referenced at run-time (in a Processing Sequence) in several ways:

- When processing is being performed for a particular Transformation Map, the attributes of the Transformation Map can be accessed through qualified forms of the \*TRANSFORM built-in variable and used as variable inputs to the Activities and Transformation Maps to be executed. Refer to Transformation Map (\*TRANSFORM) Built-in Variable Qualifiers for more information.
- In order to access the Transformation Map attributes using the qualified forms of the \*TRANSFORM variable you must first explicitly assign a value to the \*TRANSFORM built-in variable. The value you assign must be the identifier of a Transformation Map defined in the LANSA Composer system. The most common case is to assign the identifier of a Transformation Map that is linked to a Trading Partner. Your Processing Sequence might obtain the Transformation Map identifier to assign to \*TRANSFORM in one of these ways:
  - 1. Using the **\***TRADINGPARTNER.MAP.<map type> built-in variable.

Refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers for more information about the \*TRADINGPARTNER.MAP. <map type> form of the \*TRADINGPARTNER built-in variable.

- 2. 2. From the result of running the supplied FIND\_TPMAP activity. Refer to FIND\_TPMAP for information on this activity.
- 3. By "hard-coding" the identifier of a known Transformation Map.

In order to successfully use the \*TRANSFORM built-in variable and its qualified forms, a valid value must have been assigned to it in one of the above ways.

# Transformation Map (\*TRANSFORM) Built-in Variable Qualifiers

The \*TRANSFORM built-in variable may be extended with a range of qualifiers to access the attributes of the Transformation Map. The available qualified forms of the \*TRANSFORM built-in variable may be displayed and accessed by expanding the corresponding items listed on the *Built-ins* tab in the *Processing sequence editor*.

**Note:** While the \*TRANSFORM built-in variable is writeable, the qualified forms that provide access to the attributes of the current transformation map are not writeable.

### **Transformation Map Attributes**

The base attributes for the current transformation map may be accessed using the following qualifiers with the \*TRANSFORM variable:

Qualifier	Description
.II	The unique internal identifier of the Transformation Map. (The internal identifier is a 32 character value uniquely assigned by LANSA Composer. You can see the unique identifier for a Transformation Map on the <i>Audit</i> tab.)
.ID	The identifier or name of the Transformation Map definition This is the value returned for the *TRANSFORM built-in variable if no qualifier is specified. In other words *TRANSFORM yields the same value as *TRANSFORM.ID.
.STATUS	The status code (A=active, I=inactive) of the current Transformation Map.
.TYPE	The map type code for the Transformation Map definition.

### **Transformation Map Data Interchange Attributes**

The data interchange attributes for the current Transformation Map may be

Qualifier	Description
.EDI.DIR	The direction (I = inbound, O = outbound) of the current transformation map
.EDI.EXP	The transaction data export processor for the current transformation map
.EDI.IMP	The transaction data import processor for the current transformation map
.EDI.TID	The transaction ID associated with the current transformation map

accessed using the following qualifiers with the \*TRANSFORM variable:

Note: The use and values of these attributes is subject to your own circumstances and conventions. LANSA Composer does not enforce the entry of the fields corresponding to these attributes, nor does it validate any values entered.

### Using the Transaction Document Built-in Variables

Transaction document information may be referenced at run-time (in a Processing Sequence) in several ways:

- When processing is being performed for a particular transaction document, the attributes of the transaction document can be accessed through qualified forms of the \*TXDOC built-in variable. Refer to Transaction Document (\*TXDOC) Built-in Variable Qualifiers for more information.
- In order to access the transaction document attributes using the qualified forms of the \*TXDOC variable you must first explicitly assign a value to the \*TXDOC built-in variable. The value you assign must be the identifier (number) of a transaction document envelope defined in the LANSA Composer transaction document register.

### Transaction Document (\*TXDOC) Built-in Variable Qualifiers

The \*TXDOC built-in variable may be extended with a range of qualifiers to access the attributes of the transaction document. The available qualified forms of the \*TXDOC built-in variable may be displayed and accessed by expanding the corresponding items listed on the *Built-ins* tab in the *Processing sequence editor*.

<b>Note:</b> While the *TXDOC built-in variable is writeable, the qualified forms that provide access to the attributes of the current transaction document are not writeable.		
Qualifier	Description	
.ID	The identifier or number of the transaction document envelope.	
	This is the value returned for the *TXDOC built-in variable if no qualifier is specified. In other words *TXDOC yields the same value as *TXDOC.ID.	
.PID	The identifier for the parent transaction document (if any).	
.RID	The identifier for the document to which this is the response (if any).	
.DIR	The direction (I, O) for the transaction document.	
.TPID	The exchange trading partner identifier for the transaction document.	
.DATETIME_UTC	The exchange date and time for the transaction document as Universal Coordinated Time (UTC).	
.DATETIME_LOCAL	The exchange date and time for the transaction document as local time.	
.RESULT	The result code from processing the transaction document.	
.STATUS	The current status code for the transaction document.	

.STATUSTEXT	The current status description for the transaction document.
.DOCPATH	The last recorded file location for the transaction document.
.DOCORIGINAL	The original file location for the transaction document.
.DOCSIZE	The transaction document file size in bytes.
.DOCTYPE	The document type identifier for the transaction document.
.CONTENTTYPE	The content type for the transaction document.

### **3.1.4 Processing Directives**

As well as defining when and how to run Activities and Transformation Maps, Processing Sequence Directives may be used to orchestrate their execution by the use of loops, conditioning and other constructs.

The *Palette* tab in the *Processing sequence editor* shows the Processing Sequence Directives that may be used.



Detailed information on the Processing Sequence Directives follows:

Loop While And Until Leave Continue Switch, Case And Otherwise If, ElseIf And ElseActivity And TransformProcessing SequenceCatchAssignSuspend and TerminateComment

### Loop

The Loop Processing Directive executes the block of its child items iteratively for each instance of the list variable named for it. Refer to 3.1.2 Variables and Lists for more information about the use of list variables.

-Details:	
List name	filepath
Loop variable	thisfile
Index variable	

You can specify the following details for a Loop Directive:

- List Required. The *List name* specifies the name of the list variable associated with this loop. When the loop is executed it will iterate once for each contiguous indexed instance of the specified variable starting at index number 1.
- **Loop** Optional. If specified, the *Loop variable* specifies the name of a variable that will receive the indexed value of the list variable upon each iteration of the loop.
- **Index** Optional. If specified, the *Index variable* specifies the name of a variable that will receive the index value for each iteration of the loop.

# While And Until

The While and Until Processing Directives execute the block of their child items iteratively subject to the specified condition.

The While Directive tests the condition at the start of the block. If the condition is not true when the While Directive is encountered, the block is not executed.

The Until Directive tests the condition at the end of the block. Therefore, when it is reached, the Until block will always execute at least once.

You can specify the following details for a While or Until Directive:

-Details:	
Condition:	
&filename *ne "	
Index variable	my_while_index
Maximum iterations	999 🚖

- **Condition** Required. Specifies the condition to be tested for this While or Until block. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.
- Index<br/>variableOptional. If specified, the *Index variable* specifies the name of a<br/>variable that will receive the index value for each iteration of the<br/>While or Until block.
- Maximum Iterations Optional. This value is intended to prevent unintentional "runaway" loops. If the value specified is greater than zero, the While or Until block stops executing once the specified number of iterations have been processed, irrespective of the conditioning expression.

The default value is 999. It is not expected that Processing Sequences will usually use long-running loops. When a While or Until block ends prematurely due to reaching the maximum iterations an error is raised in the Processing Sequence.

### Leave

The Leave Processing Directive causes processing to immediately leave the closest enclosing Loop, While or Until Directive or Iterator Activity. The enclosing loop terminates irrespective of any list or condition to which it is otherwise subject and processing continues with the next logical directive after the end of the enclosing Loop, While or Until block.

You can specify the following details for a Leave Directive:

í	-Details:
	Condition:
	&acknowledgements = YES'
l	

**Condition** Optional. Leave is unconditional if no condition is specified. Alternatively you can specify a condition that must be satisfied for the Leave Directive to be effective. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.

### Continue

The Continue Processing Directive causes processing to immediately resume with the next iteration of the closest enclosing Loop, While or Until Directive or Iterator Activity without executing further directives contained within the block for the current iteration.

You can specify the following details for a Continue Directive:

-Details:	
Condition:	
&acknowledgements = 'YES'	

**Condition** Optional. Continue is unconditional if no condition is specified. Alternatively you can specify a condition that must be satisfied for the Continue Directive to be effective. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.

### Switch, Case And Otherwise

The Switch, Case and Otherwise Directives can be used together to construct one or more blocks of items to be executed conditionally. For any one Switch block, one and only one of the Case or Otherwise blocks will be executed. The block executed will be the first Case block whose associated condition is true or the Otherwise block (if specified) if none of the Case conditions are true.

There are no additional details to complete in the detailer window for the Switch and Otherwise Directives.

For the Case Directive, you must specify the condition that is to be tested in the detailer window.

-Details:
Condition:
&acknowledgements = 'YES'

**Condition** Required. Specifies the condition to be tested for this Case block. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.

# If, ElseIf And Else

The If, ElseIf and Else Directives can be used together to construct one or more blocks of items to be executed conditionally. For any one If Directive, one and only one of the If, ElseIf or Else blocks will be executed. The block executed will be the first If or ElseIf block whose associated condition is true or the Else block (if specified) if none of the conditions are true.

There are no additional details to complete in the detailer window for the Else Directive.

For the If and ElseIf Directives, you must specify the condition that is to be tested in the detailer window.

-Details:
Condition:
&acknowledgements = 'YES'

**Condition** Required. Specifies the condition to be tested for this If or ElseIf block. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.

# **Activity And Transform**

The Activity and Transform Processing Directives execute the Activity or Transformation Map specified for the item.

You must identify the specific Activity or Transformation Map that is to be executed. Usually you do this by dragging items from the *Activities* or *Transformations* tab directly into the Processing Sequence. However, you may also change the Activity or Transformation Map for an existing Processing Sequence item by dragging an item from the *Activities* or *Transformations* tab into the detailer window for the item.

The *Details* tab of the detailer window identifies the Activity or Transformation Map that is specified for this item.

Details Parameters			
Activity ID	FTP_INBOUND		
Activity description	FTP Get files from remote host		

The *Parameters* tab of the detailer window shows the parameters received or returned by the Activity or Transformation Map.

	Details Parameters					
				Variable or value for parameter	Parameter Name	Parameter description
ľ	1	-	Inbound	'DEM_FTP_IN'	FTPCONFIG	FTP Configuration ID
l	2		Inbound		REMOTEDIRECTI	Override Remote Directory to GET file f
	3		Inbound		LOCALDIRECTOF	Override Local Directory into which file
	4		Outbound		FILELIST	List of file retrieved - file name
	5		Outbound		FILEPATH	List of file retrieved - full path & name
	<	_				>

\*

An asterisk in the second column denotes an inbound parameter that is flagged as being required. This means that a variable or value must be specified for it if its value has not already been set by a preceding Directive such as an Assign or another Activity.

InboundOutbound The second column of the list shows whether the associated parameter is inbound (received by the Activity or Transformation Map), outbound (returned by the Activity or Transformation Map) or both. An asterisk

	in the second column denotes an inbound parameter that is flagged as being required.			
Variable or value for parameter	Optional. Specifies the name of a variable or a numeric or alphanumeric literal that is to be used for the parameter. Refer to 3.1.1 Activities and Transformation Maps for more information on specifying values for Activity or Transformation Map parameters,			
Parameter Name Parameter Description	These columns show the parameter names and parameter descriptions as specified in the Activity or Transformation Map definition. The parameter names shown are added to the variable pool for the Processing Sequence for use in this and other Processing Sequence items.			

### **Processing Sequence**

The Processing Sequence Processing Directive executes the Processing Sequence specified for the item. In this way, one Processing Sequence can execute another Processing Sequence.

Note: While LANSA Composer imposes no inherent limit on the level of nesting of Processing Sequences that can be achieved in this way, it is strongly recommended that you do not nest Processing Sequences to a depth greater than two or three.

The LANSA run-time environment used by LANSA Composer does limit the degree of nesting either within a single Processing Sequence and/or when nesting Processing Sequences. In any event, excessive degrees of nesting may adversely impact the performance of your business process integration solutions.

You can identify the specific Processing Sequence that is to be executed. Usually you do this by dragging items from the *Processing Sequences* tab directly into the Processing Sequence. However, you may also change the Processing Sequence for an existing Processing Sequence item by dragging an item from the *Processing Sequences* tab into the detailer window for the item.

If you wish, you can specify a variable name whose value will be used at runtime to determine the Processing Sequence to execute. If you specify the Variable for Processing Sequence, it will override the specific Processing Sequence identified above it. (However, you may still wish to provide a specific Processing Sequence to be used as a model to allow you to enter parameter values for the Processing Sequence).

The *Details* tab of the detailer window identifies the Processing Sequence that is specified for this item and, optionally, the name of the variable that will contain the Processing Sequence name or identity at run-time..

Details Parameters				
Processing Sequence ID	TXDOC_IEDI020			
Processing Sequence Description	EDI transaction document process			
Variable for processing sequence:				

The *Parameters* tab of the detailer window shows the parameters received or returned by the Processing Sequence.

Note: When the nested Processing Sequence runs, it has its own independent variable pool. Variables in the containing Processing Sequence are not accessible to the nested Processing Sequence, nor vice-versa. Communication of variables is only possible through the Processing Sequence parameters.

Details Parameters					
			Variable or value for parameter	Parameter Name	Parameter description
1		Inbound	*tradingpartner	IN_TP	Trading partner id
2		Inbound	&thisdoc	IN_TXDOC	Transaction document file path
3		Inbound	&thisdoctype	IN_TXDOCTYPE	Transaction document type
4		Inbound	&thisdocnumber	IN_TXDOCNUMB	Transaction document envelope nu
				Þ	

- An asterisk in the second column denotes an inbound parameter that is flagged as being required. This means that a variable or value must be specified for it if its value has not already been set by a preceding Directive such as an Assign or an Activity.
- Inbound The second column of the list shows whether the associatedOutbound parameter is inbound (received by the Processing Sequence), outbound (returned by the Processing Sequence) or both.
- Variable or value for parameter
   Optional. Specifies the name of a variable or a numeric or alphanumeric literal that is to be used for the parameter. Refer to 3.1.1 Activities and Transformation Maps for more information on specifying values for Processing Sequence parameters,

ParameterThese columns show the parameter names and parameterNamedescriptions as specified in the Processing Sequence definition.ParameterThe parameter names shown are added to the variable pool forDescriptionthe Processing Sequence for use in this and other Processing<br/>Sequence items.
# Catch

You can place one or more Catch Processing Directives after an Activity, Transformation Map or Processing Sequence directive. For each Activity, Transformation Map or Processing Sequence directive, a maximum of one and only one of the Catch blocks (if specified) will be executed. The block executed will be the first that matches the result code from the Activity, Transformation Map or Processing Sequence directive.

Each Catch Directive can itself have child items that are executed when the Catch item is activated. If a Catch block is activated, any error condition is handled and the activation of the Catch block will prevent the error from ending the Processing Sequence. If you wish, you may explicitly end the Processing Sequence (with or without error) by including a Terminate Directive in the Catch block.

You can specify the following details for Catch Directives:



Result Specifies the condition that will activate the CATCH directive. If you select 'Errors', the CATCH directive will be activated when the Activity return code signifies an error ('ER'). If you select 'OK', the CATCH directive will be activated when the Activity ends without error. (The latter case is less usual, but it allows you to specify alternate error handling and normal completion actions.)

## Assign

The Assign Processing Directive assigns a value to a variable in the variable pool for the Processing Sequence. In many cases variable and variable list values are set by running Activities, but this directive permits a value to be directly assigned. Refer to 3.1.2 Variables for more information.

Details:
Expression:
&thisfile = &filepath(&fileindex)

**Expression** Required. Specifies the assignment expression. Refer to Assignment Expressions for information about the syntax of assignment expressions.

# **Suspend and Terminate**

The Suspend and Terminate Processing Directives terminate the Processing Sequence. Suspend terminates the Processing Sequence with an error ('ER') condition, such that it can subsequently be restarted. The Terminate directive terminates the Processing Sequence with the specified result code (but, in any event, the Processing Sequence cannot be restarted). Either directive would typically be used subject to a condition specified on the directive itself or in an enclosing conditional block such as Case, Otherwise, If, Elseif or Else.

You can specify the following details for Suspend and Terminate Directives:

Details:			
Condition:			
&MAPCOUNT = 0			
Result code:			
Errors	<b>*</b>		
Message text:			
Quod erat demonstrandum			
Perform variable substitution?			

- **Condition** Optional. The directives are unconditional if no condition is specified. Alternatively you can specify a condition that must be satisfied for the Directive to be effective. Refer to Conditioning Expressions for information about the syntax of conditioning expressions.
- **Result code** (Applies to Terminate directive only) Specifies the result code with which the Processing Sequence will end. Select 'OK' if the Processing Sequence should end normally or 'Errors' if the Processing Sequence should end with an error ('ER') condition.
- MessageOptional. You may specify message text that will appear in the<br/>Processing Sequence log if/when the Suspend or Terminate<br/>item is executed. You may specify a short message and/or<br/>extended (detailed) message text.

# Perform If this box is checked, then the message text specified in the preceding fields may contain embedded references to processing sequence variables and/or built-in variables which

will be replaced at run-time with the corresponding variable values.

For example, if you specify the string "&MYCOUNT files processed from &MYFOLDER(&MYINDEX) at \*now\_local", the variable and built-in variable references &MYCOUNT, &MYFOLDER(&MYINDEX) and \*now\_local will be replaced by the values of the variables that they represent.

In most cases, a reference to a processing sequence variable or built-invariable must be followed by a space (or certain special characters or the end of the string) in order for the reference to be correctly identified and substituted.

If two ampersands (&&) or two asterisks (\*\*) appear together in the string, they are reduced to a single ampersand or asterisk and not considered for substitution.

# Comment

The Comment Processing Directive allows you to annotate the processing sequence with comments. Comment Directives are not executed and have no other effect on the processing sequence definition or execution.

When you add or select a Comment Directive in your processing sequence, the detailer window provides an edit box in which you can enter or edit one or more lines of comments.

Comment:			
Loop for each trading partner			

# **Assignment Expressions**

The assignment expression used in the ASSIGN directive consists of the following parts:

LeftThe name of a variable or built-in variable that receives the valueOperandspecified by the right operand.

To specify a variable, precede the variable name with an ampersand (&) – for example &VARIABLE1. To specify an indexed variable, specify a variable or numeric literal that specifies the index value in parentheses immediately following the variable name – for example &VARIABLE1(&INDEX1).

**Note:** The variable specified by the left operand does not have to have been previously defined in the variable pool. If it does not exist, the Assign Processing Directive implicitly defines it.

To specify a built-in variable, enter the name, including the asterisk, as shown on the *Built-ins* tab – do not precede a built-in variable name with ampersand. Note only certain built-in variables are valid as the target of an assignment expression. Refer to 3.1.3 Built-in Variables for information.

Operator An assignment operator used to assign a value to the variable specified by the left operand. It may be one of the following:
 = assigns the value of the right operand to the variable specified by

the left operand

+= adds the numeric value of the right operand to the numeric value of the variable specified by the left operand

-= subtracts the numeric value of the right operand from the numeric value of the variable specified by the left operand

When += or -= assignment operators are used the values of the left and right operands are evaluated as numeric if possible. If either operand does not contain a numeric value, then zero is used as the value for that operand.

Right The value that is assigned to the variable specified by the left

Operand operand. The value can be specified in one of two ways:

1. A numeric or alphanumeric literal value. If you enter an alphanumeric literal value you must surround it with quote marks. The quote marks do not become part of the value that is assigned. If you need to embed quote marks in the value, you double-up the embedded quote marks.

2. The name of a variable or built-in variable that supplies the value for the assignment. To specify a variable, precede the variable name with an ampersand (&) – for example &VARIABLE1. To specify a built-in variable, enter the name, including the asterisk, as shown on the *Built-ins* tab – do not precede a built-in variable name with ampersand. To specify an indexed variable, specify a variable or numeric literal that specifies the index value in parentheses immediately following the variable name – for example &VARIABLE1(&INDEX1).

# Examples

This expression assigns the numeric value 3 to the variable named MYNUMBER:

&MYNUMBER = 3

This expression adds 1 to the numeric value of the variable named MYNUMBER:

## &MYNUMBER += 1

This expression adds the numeric value of the variable named OTHERNUMBER to the numeric value of the variable named MYNUMBER:

#### &MYNUMBER += &OTHERNUMBER

This expression assigns the alphanumeric value ABC to the variable named MYSTRING:

&MYSTRING = 'ABC'

These three assignment expressions populate the first three instances of the MYLIST variable list with the first three letters of the alphabet:

&MYLIST(1) = 'A' &MYLIST(2) = 'B' &MYLIST(3) = 'C' This expression assigns the value of the third instance of the MYLIST variable list to the MYLETTER variable:

#### &MYLETTER = &MYLIST(3)

This expression is similar but the MYINDEX variable specifies which instance of the MYLIST variable list to use:

# &MYLETTER = &MYLIST(&MYINDEX)

This expression assigns trading partner identifier ATLAS to the \*TRADINGPARTNER built-in variable:

## \*TRADINGPARTNER = 'ATLAS'

This expression assigns the identifier of the FTP Inbound Configuration linked to the current trading partner to the variable MYFTPCONFIG:

## &MYFTPCONFIG = \*TRADINGPARTNER.FTPIN

# **Conditioning Expressions**

The conditioning expression used in WHILE, UNTIL, CASE, IF and ELSEIF directives consists of the following parts:

LeftThe name of a variable or Built-in variable whose value is tested byOperandthe expression.

To specify a variable, precede the variable name with an ampersand (&) – for example &VARIABLE1. To specify an indexed variable, specify a variable or numeric literal that specifies the index value in parentheses immediately following the variable name – for example &VARIABLE1(&INDEX1).

To specify a built-in variable, enter the name, including the asterisk, as shown on the *Built-ins* tab – do not precede a built-in variable name with ampersand.

- Operator A comparison operator used to compare the value of the left and right operands. It may be one of the following:
  - \*EQ = Equal
  - \*NE <> Not equal
  - \*GT > Greater than
  - \*GE >= Greater than or equal
  - \*LT < Less than
  - \*LE <= Less than or equal

RightThe value that is compared to the variable specified by the leftOperandoperand. The value can be specified in one of two ways:

- 1. A numeric or alphanumeric literal value. If you enter an alphanumeric literal value you must surround it with quote marks. The quote marks do not become part of the value used for the comparison. If you need to embed quote marks in the value, you should double-up the embedded quote marks.
- The name of a variable or built-in variable that supplies the value used for the comparison. To specify a variable, precede the variable name with an ampersand (&) for example &VARIABLE1. To specify a built-in variable, enter the name,

including the asterisk, as shown on the *Built-ins* tab – do not precede a built-in variable name with ampersand. To specify an indexed variable, specify a variable or numeric literal that specifies the index value in parentheses immediately following the variable name – for example &VARIABLE1(&INDEX1).

# Examples

This expression is true when the value of the variable named MYNUMBER is greater than 3:

# &MYNUMBER \*GT 3

This expression is true when the value of the variable named MYSTRING: is ABC:

## &MYSTRING \*EQ 'ABC'

This expression is true when the value of the second instance of the MYLIST variable list is B:

## &MYLIST(2) = 'B'

This expression is true when the value of the MYLETTER variable is not equal to the value of the third instance of the MYLIST variable list:

# &MYLETTER <> &MYLIST(3)

This expression is similar but the MYINDEX variable specifies which instance of the MYLIST variable list to use for the comparison:

## &MYLETTER <> &MYLIST(&MYINDEX)

This expression is true when the value of the \*TRADINGPARTNER built-in variable is ATLAS:

## \*TRADINGPARTNER = 'ATLAS'

#### 3.1.5 Parameters

Processing sequences can be defined to receive parameters that become variables in the variable pool. The variables representing the parameters can be used like any other variable as input to an Activity or Transformation Map.

Parameters are defined in the *Parameters* tab of the *Processing sequence editor* window. Refer to 3.4.5 Edit Processing Sequence Parameters for more information.



# 3.2 Work With Processing Sequences

Using LANSA Composer you can create, maintain, run and delete any number of Processing Sequences.

To work with Processing Sequences, expand *Definitions* in the *Navigator* and click *Processing sequences*. To find out how to locate and select Processing Sequences to work with, refer to:

#### Locating and selecting items in the Instance Lists

For information on common tasks associated with Processing Sequences (such as creating, copying, deleting and printing definitions) refer to:

Working with definition items

For information on tasks associated specifically with Processing Sequences, refer to the following headings:

Display the run history of a Processing Sequence Display, Delete or Restore Prior Versions of a Processing Sequence

#### Display the run history of a Processing Sequence

To display the run history of a Processing Sequence, select the required item in the Processing Sequence list and click the *Run history* tab. A list of recorded prior runs of the Processing Sequence will be displayed.

Select an item and click the *View* button to display the Processing Sequence log or the *Print* button to print the Processing Sequence log for the run.

If a prior Processing Sequence run has ended in error, you may be able to use the *Restart* button to restart it from the point of failure. Refer to 3.6 Restart a Processing Sequence Run for more information.

Note: Processing Sequence run history can be purged using the Database Housekeeping function. The run history for Processing Sequence runs that have been purged is no longer available.

#### Display, Delete or Restore Prior Versions of a Processing Sequence

To work with prior versions of a Processing Sequence, select the item you wish to delete in the Processing Sequence list and click the *Version history* tab. A list of available prior versions of the Processing Sequence will be displayed. Refer to 3.3 Work With Processing Sequence Version History for more information.

# 3.3 Work With Processing Sequence Version History

LANSA Composer can store multiple versions of the definition of a Processing Sequence. This permits the flexibility to restore and run an earlier version of the Processing Sequence or simply to review how the definition of a Processing Sequence has developed.

Each time you save a Processing Sequence definition using the Processing Sequence Editor, the previous version of the Processing Sequence is archived (subject to the corresponding System Setting). The archived versions contain the complete executable definition of a Processing Sequence, but does not include:

- The Processing Sequence attachments or notes
- Any Activities, Transformation Maps, Trading Partners, Configurations or other items that do not form part of the Processing Sequence definition but that might be referenced by it.

To review the available prior versions of a Processing Sequence:

- 1. In the Navigator, expand Definitions and select Processing sequences.
- 2. Use the filter and the items list to locate and select the Processing Sequence that you wish to review.
- 3. Click the Version history tab.

A list of available prior versions of the Processing Sequence will be displayed. You can then:

View a prior version of the Processing Sequence

Delete a prior version of the Processing Sequence

Restore a prior version of the Processing Sequence



#### View a prior version of the Processing Sequence

To view the definition of a prior version of the Processing Sequence, select the version you wish to view and click the *View* button. LANSA Composer will open the prior version of the Processing Sequence definition in the *Processing sequence editor* window.

Note: the *Processing sequence editor* window opens in *read-only* mode - you cannot make any changes to the definition.

#### Delete a prior version of the Processing Sequence

To delete one or more prior versions of a Processing Sequence, select the version(s) you wish to delete and click the *Delete* button. A message box will ask you to confirm the deletion. Click the *Yes* button in the message box to complete the deletion.

#### **Restore a prior version of the Processing Sequence**

To restore a prior version of a Processing Sequence, select the version you wish to restore and click the *Restore* button. A message box will ask you to confirm the restore. Click the *Yes* button in the message box to complete the restore.

When you restore a prior version, the definition held in that prior version becomes the current Processing Sequence definition and the current definition replaces the selected prior version in the version history (you can subsequently restore it if you wish).

Note that the Processing Sequence attachments and notes are not held as part of the version history and are not affected by restoring a prior version. Nor are activities, transformation maps, trading partners, configurations or other items that do not form part of the Processing Sequence definition but that might be referenced by it.

# **3.4 Use the Processing Sequence Editor**

You use the *Processing sequence editor* to define the way in which Activities and Transformations maps are to be combined and executed in order to complete a business process.

To edit a Processing Sequence in the *Processing sequence editor*:

- 1. In the *Navigator*, expand *Definitions* and select *Processing sequences*.
- 2. Use the filter and the items list to locate and select the Processing Sequence that you wish to edit.
- 3. Press the *Edit* button in the *Details* tab to open the Processing Sequence in the *Processing sequence editor* window.

#### Also see

3.4.1 Parts of the Editor Window

# 3.4.1 Parts of the Editor Window

The *Processing sequence editor* window has these parts:



You can resize the parts by clicking and dragging in the vertical or horizontal space between the major parts, as shown in the illustration.

#### 1. Resources

The tabs in the resources window provide easy access to the components you will use to define your Processing Sequence such as Processing Directives, Activities and Transformation Maps.

Refer to 3.4.2 The Resources Tabs for more information.

#### 2. Information

The tabs in the information window provide:

- access to help or notes for the Processing Sequence directive, Activity, Processing Sequence, Transformation Map or Configuration last selected in the resources window
- validation errors updated as you edit your Processing Sequence

Refer to 3.4.3 The Information Tabs for more information.

# 3. Processing sequence editor

The Processing Sequence editor window visualizes your Processing Sequence and allows you to edit it by, for example, dragging and dropping items such as Activities from the resources window.

Refer to 3.4.4 Edit Processing Sequence Details for more information.

# 4. Detailer

The detailer window displays and allows you to edit details for the item selected in the Processing Sequence editor window. For example you can edit the values used for an Activities parameter.

Refer to Review or Revise Item Details for more information.

# **Processing Sequence Parameters Editor**

The Processing Sequence parameters editor window displays and allows you to edit the parameters that can be passed to the Processing Sequence. It is not shown in the illustration, but can be displayed by clicking the *Parameters* tab.

Refer to 3.4.5 Edit Processing Sequence Parameters for more information.

# **Processing Sequence Notes Editor**

The Processing Sequence notes editor window displays and allows you to revise the notes associated with the Processing Sequence. It is not shown in the illustration, but can be displayed by clicking the *Notes* tab.

Refer to 3.4.6 Edit Processing Sequence Notes for more information.

# 5. Menu and Toolbar

The drop-down menus and the toolbar buttons provide access to the major editing commands you use to define and modify your Processing Sequence.

#### 3.4.2 The Resources Tabs

The tabs in the resources window provide easy access to the components you will use to define your Processing Sequence. You can drag and drop items from these tabs into the Processing Sequence editor windows when you want to add or reference the item in your Processing Sequence. The following resources tabs are available:

Palette tab Activities tab Processing Sequences tab Transformations tab Configurations tab Variables tab Built-ins tab

# Palette tab

The *Palette* tab shows icons for each Processing Sequence Directive you can insert in your Processing Sequence.



To add a Directive to your Processing Sequence, click the icon for the required item and drag it over to the Processing Sequence editor window. When you release the mouse, the item will be inserted at the highlighted location. You may then need to complete details for the item in the *Detailer* window.

You can also add a Directive to the Processing Sequence by double clicking the icon for the required Directive type. The new item will be inserted at the current selection.

Refer to 3.1.4 Processing Directives for more information about the Processing Sequence Directives you can use.

# Activities tab

The *Activities* tab provides access to lists of the Activities defined in LANSA Composer. Refer to Activities for more information about defining Activities.



You can type a search term in the *Search* box at the top of the *Activities* tab to search for Activities that contain the search term in their name or description.

Alternatively, you can expand the tree nodes on the *Activities* tab to list Activities by group, or by alphabetical grouping of the Activity name.

**Note:** Activity groups are user-defined. LANSA Composer is supplied with pre-defined Activity groups and the supplied Activities are assigned to these Activity groups. You may define your own Activity groups in addition to or in place of those supplied with LANSA Composer. Refer to Code maintenance for information on defining activity groups.

Click an Activity to display its notes in the Assistant tab.

To add an Activity to your Processing Sequence, click the required Activity and drag it over to the Processing Sequence editor window. When you release the mouse, the Activity item will be inserted at the highlighted location. You may then need to complete details for the Activity item in the *Detailer* window – in particular, you will need to specify variables or values for the Activity parameters, if any. Refer to 3.1.1 Activities and Transformation Maps for more information.

You can also add an Activity to the Processing Sequence by double clicking it. The new Activity item will be inserted at the current selection.

# **Processing Sequences tab**

The *Processing Sequences* tab provides access to lists of the Processing Sequences defined in LANSA Composer.



You can type a search term in the *Search* box at the top of the *Processing Sequences* tab to search for Processing Sequences that contain the search term in their name or description.

Alternatively, you can expand the tree nodes on the *Processing Sequences* tab to list Processing Sequences by alphabetical grouping of the Processing Sequence name.

Click a Processing Sequence to display its notes in the Assistant tab.

To add a Processing Sequence, click the required item and drag it over the Processing Sequence editor window. When you release the mouse, the Processing Sequence item will be inserted at the highlighted location. You may then need to complete details for the Processing Sequence item in the *Detailer* window – in particular, you will need to specify variables or values for the Processing Sequence parameters, if any. Refer to 3.1.1 Activities and Transformation Maps for more information.

You can also add a Processing Sequence by double clicking it. The new Processing Sequence item will be inserted at the current selection.

# Transformations tab

The *Transformations* tab provides access to lists of the Transformation Maps defined in LANSA Composer. Refer to Transformation Maps for information about defining Transformation Maps.



You can type a search term in the *Search* box at the top of the *Transformations* tab to search for Transformation Maps that contain the search term in their name or description.

Alternatively, you can expand the tree nodes on the *Transformations* tab to list Transformation Maps by type, or by alphabetical grouping of the Transformation Map name.

**Note:** Transformation map types are user-defined. The dropdown list will contain Transformation Map types only if you have defined types in LANSA Composer.

Click a Transformation Map to display its notes in the Assistant tab.

To add a Transformation to your Processing Sequence, click the required Transformation Map and drag it over to the Processing Sequence editor window. When you release the mouse, the Transformation Map item will be inserted at the highlighted location. You may then need to complete details for the Transformation Map item in the *Detailer* window – in particular, you will need to specify variables or values for the Transformation Map parameters, if any. Refer to 3.1.1 Activities and Transformation Maps for more information.

You can also add a Transformation Map to the Processing Sequence by double clicking it. The new Transformation Map item will be inserted at the current selection.

**Note:** You can also execute a Transformation Map using the supplied TRANSFORM activity. This is intended for cases where you do not know the identifier of the Transformation Map to execute until runtime. Refer to **TRANSFORM** for information on this activity.

# **Configurations tab**

The *Configurations* tab provides access to lists of the transport and/or database Configurations available for you to reference in your Processing Sequence. Refer to Configurations for more information about Configurations.



You can type a search term in the *Search* box at the top of the *Configurations* tab to search for Configurations that contain the search term in their name or description.

Alternatively, you can expand the tree nodes on the *Configurations* tab to list Configurations by group, or by alphabetical grouping of the Configuration name.

Click a Configuration to display its notes in the Assistant tab.

You can drag Configurations to places that specify a configuration name – for example, to Activity parameters that specify a configuration name. The FTP\_INBOUND shipped Activity, for example, receives a configuration name in its FTPCONFIG parameter.

You can also drag Configurations directly over the Processing Sequence editor window. When you release the mouse, a default Activity (according to the Configuration type) will be inserted at the highlighted location. For example, if you drag and drop an FTP inbound configuration, LANSA Composer will insert the FTP\_INBOUND activity and set the FTPCONFIG parameter value to the name of the FTP Configuration.

# Variables tab

The *Variables* tab provides access to variables available in the variable pool for the Processing Sequence. Variables are added to or removed from the pool and the list on the *Variables* tab as Activities, Transformation Maps and other references to variables in Processing Sequence Directives are added, removed or modified.

You can reference the variables listed on the *Variables* tab in your Processing Sequence. Refer to 3.1.2 Variables for more information.



You can drag variables from the *Variables* tab to places where you can enter a variable name including:

- Activity and Transformation Map parameters
- Loop Directive field definitions
- conditional and assignment expressions
- the Processing Sequence Parameters Editor.

Expand a variable item in the *Variables* tab to see references to the variable. You can double click the variable reference to select the corresponding item in the Processing Sequence editor.

# **Built-ins tab**

The *Built-ins* tab provides access to built-in variables defined by LANSA Composer that you can reference in your Processing Sequence. Refer to 3.1.3 Built-in Variables for more information.



The available built-in variables are shown in a tree, with some of the available variables functionally grouped together. Click the '+' sign next to a group name to see the available built-in variables in that group.

For more information about a built-in variable, pause the mouse pointer over the variable until a pop-up "hint" is momentarily displayed. Alternatively, click the variable name to its description in the status bar.

You can drag built-in variables from the *Built-ins* tab to places where you can enter a variable name including:

- Activity and Transformation Map parameters
- conditional and assignment expressions

You cannot drag built-in variables to the *Processing Sequence Parameters Editor*. In addition, most built-in variables are not valid in cases where "write" access to the variable is required – for example as an output parameter for an Activity or on the left-hand side of an assignment expression.

## **3.4.3 The Information Tabs**

The tabs in the information window provide information to help you as you define your Processing Sequence. The following information tabs are available:

Assistant tab Errors tab

#### Assistant tab

The *Assistant* tab displays help or notes for the Processing Sequence directive, Activity, Processing Sequence, Transformation Map or Configuration last selected in the resources window. This information can help you in choosing and correctly using the items you use in your Processing Sequence.



Notes are provided for all supplied Activities. It is your responsibility as a LANSA Composer designer to provide notes for Activities, Transformation Maps, Configurations and Processing Sequences that you define. Refer to Define or Review Notes for more information.

# Errors tab

The *Errors* tab displays validation warnings and errors. The warnings and errors are updated dynamically as you edit your Processing Sequence.



Information and Warnings are displayed with blue and orange icons and may not require any action from you.

Errors are displayed with a red icon and do require action, though you may not need to action them immediately. Note that you can save a Processing Sequence with errors. The errors will be displayed again when you reload the Processing Sequence into the editor.

If you attempt to run a Processing Sequence that was saved with errors, it may not run as expected or it may not run to completion.

It is normal for errors to appear and disappear in the course of editing your Processing Sequence. For example, immediately after adding a Loop Processing Directive you will see an error like the third one shown. This simply indicates that the definition is incomplete. The error will disappear when you complete the associated definition.

**Note:** You can double click on an error to select the corresponding item in the Processing Sequence editor.

# **3.4.4 Edit Processing Sequence Details**

The *Processing sequence editor* visualizes the Processing Sequence as a tree of Activities, Transformation Maps and Processing Sequence Directives. You can add comments by using the Comment Processing Sequence Directive.

		*** Beginning of processing sequence ***
÷	Activity	
	Name:	FTP_INBOUND
	Description:	FTP Get files from remote host
÷	Activity	
	Name:	DEMO_BATCHNUM
	Description:	Generate batch number
÷	Loop	
	List:	FILEPATH
	Loop variable:	THISFILE
ė.	ᡖ Transform	
	Name:	DEM_ORDER_DB2
	Description:	order.xml to DB2
÷	Activity	
	Name:	DEM_ORDER_RCV
	Description:	Process received orders
□ (7)	lf	
	Condition:	&ACKNOWLEDGEMENTS = 'YES'
Ē.	ᡖ Transform	
	Name:	DEM_DB2_ACK
	Description:	DB2 to ack.xml
Ė.	land the second	
	Name:	FTP_OUTBOUND
	Description:	FTP Put files on remote host
		*** End of processing sequence ***

The relationships in the tree represent the relationships between the items. For example, items that are indented beneath a Loop item are the items that will be executed for each iteration of the loop.

Using the *Processing sequence editor*, you can:

Add Items to the Processing Sequence

Delete Items from the Processing Sequence

Cut, Copy and Paste Items in the Processing Sequence

Re-arrange Items in the Processing Sequence

Move Items Up Or Down

Promote or Demote Items

Enable or Disable Items

Review or Revise Item Details

Customise the view in the Processing Sequence Editor

# Add Items to the Processing Sequence

You add items to the Processing Sequence by dragging them from the *Palette*, *Activities, Processing Sequences, Transformations* or *Configurations* tabs. Alternatively, you can double click items on those tabs to add them to the Processing Sequence or select an option from the *Insert* menu.

You may then need to complete details for the item in the *Detailer* window. For more information on adding items to the Processing Sequence, refer to:

Palette tab Activities tab Processing Sequences tab Transformations tab Configurations tab

**Note:** When you add items to the Processing Sequence in this way, LANSA Composer will attempt to guess the most appropriate level for the item and assign its parent accordingly. The result is that the item will be indented to that level. In many cases, LANSA Composer's guess will be correct and you will not need to re-arrange the item. If it is not correct however, you should use the features described in Rearrange Items in the Processing Sequence to arrange the item in the correct order and at the correct level.

# **Delete Items from the Processing Sequence**

To delete an item from the Processing Sequence:

- 1. Select the item or items to be deleted.
- 2. Click the *Delete* button on the toolbar or select *Delete* from the *Edit* menu.
- 3. Confirm the deletion by clicking *Yes* in the message box.

**Note:** When you delete an item, any remaining child items of that item are promoted to the level and position of the deleted item.

# Cut, Copy and Paste Items in the Processing Sequence

You can cut, copy and paste items in the Processing Sequence using the *Cut*, *Copy* and *Paste* buttons on the toolbar or by selecting the corresponding commands from the *Edit* menu.

When you cut or copy an item or items to the clipboard, the entire item definitions are copied, including any details specified in the detailer window. The item definitions remains available on the clipboard indefinitely, until this or some other application copies something else to the clipboard. You can paste copied or cut items into the *Processing Sequence Editor* for this or another Processing Sequence.

**Note:** When you cut an item, any remaining child items of that item are promoted to the level and position of the cut item.

# **Re-arrange Items in the Processing Sequence**

You can re-arrange items by moving them up or down the Processing Sequence or by promoting or demoting them.

# Move Items Up Or Down

Moving items up or down in the Processing Sequence changes the order of items in the Processing Sequence and, therefore, the order in which they will be executed when the Processing Sequence is run.

To move an item up or down the Processing Sequence:

- 1. Select the item to be moved
- 2. Click the *Move Up* for *Move Down* button on the toolbar (or select the corresponding command from the *Edit* menu) until the item is in the desired position.

OR you can also use *Alt+Up Arrow* and *Alt+Down Arrow* keyboard shortcuts.

OR, you can hold the mouse button down and drag the selected item(s) to the desired position.

**Note:** As you move items up or down the Processing Sequence, it will often be necessary for LANSA Composer to change the item's parent. This will result in the indentation level of the item changing as it moves.

# **Promote or Demote Items**

Promoting or demoting items changes their parent item – that is, the item to which they belong and upon which their execution is dependent. The visible

effect of this is to change their indentation level.

To promote or demote an item:

- 1. Select the item to be promoted or demoted.
- Click the *Promote* Promote
  Promote
  Or *Demote* Demote
  Demote
  button on the toolbar (or select the corresponding command from the *Edit* menu) until the item is at the desired level. You can also use *Alt+Left Arrow* and *Alt+Right Arrow* keyboard shortcuts.

**Note:** Sometimes, as you promote or demote items, they will move up or down as well. This can occur, for example, when you promote or demote an item that is in the middle of a group of items at the same level. LANSA Composer will need to move the item in order to preserve the levels or relationships of other items in the group.

# **Enable or Disable Items**

Enabling or disabling items leaves the definition of the item in the Processing Sequence but prevents it from being executed if the Processing Sequence is run. This is sometimes useful for temporary changes or during design and testing of a Processing Sequence.

Disable items are shown with a greyed icon and text in the processing sequence editor.

To enable or disable an item:

- 1. Select the item to be enabled or disabled.
- 2. Select *Enable* from the *Edit* menu.

Note: Disabling an item also disables any child items of that item.
### **Review or Revise Item Details**

Many Processing Sequence items have additional details that affect their operation.

For example, an Activity item must specify which Activity is to be performed, and may also require that you specify values for its parameters. Similarly a Loop item requires you to specify the list variable name for which the Loop is to be executed.

These details vary according to the type of Processing Sequence item, but in all cases they are displayed and can be revised using the detailer window below the Processing Sequence tree.

To review or revise item details:

- 1. Select the item to be reviewed the details for the item (if any) are displayed in the detailer window.
- 2. Review or revise the details as required.

Refer to 3.1.4 Processing Directives for information on completing the details that apply to each type of Processing Sequence item.

### **Customise the view in the Processing Sequence Editor**

You can change aspects of the way the *Processing sequence editor* visualizes the Processing Sequence. You may find that the options provided permit you to more clearly understand the relationships between items in your Processing Sequence.

To customise the view options, select *Options* from the *View* menu and then make your choices in the *Processing Sequence Editor Options* window.

Processing Sequer	nce Editor Options		X
Choos the eff Sh Sh Choos Choo	the view options to apply in the P fect of the options on the sample be now tree lines now seguence numbers plour code according to nesting leve Number of colours 7 1 2 3 4	rocessing Sequence Editor. Review	OK Cancel Reset
Sample:		D-t-3	Carles 1
Titem		Nest level 1	Seq
*		incst incort i	
🖃 🧄 Switch			2
/ *		Nest level 2	
🖃 🔬 Case			4
Condition	ר:	&A = &B	
2*		Nest level 3	
⊡{+} If	les-		6
Conc	aluon: *	8C = 8D	
	000	NESLIEVEL4	8
	ist:	&LIST	•
	*	Nest level 5	
	👌 While		10
	Condition:	&E < &F	
	*	Nest level 6	
	🖻 🔇 Until		12
	Condition:	&G = 10	
		Nest level 7	
	- S Iterator	FOD EACH THDEY THINK	14
	iname:	Nest level 9	····
		INCALIEVELO	

### **3.4.5 Edit Processing Sequence Parameters**

The *Parameters* tab of the *Processing sequence editor* allows you to add, delete and revise the definition of parameters that the Processing Sequence can receive.

Inbound Optional CUSTOMER_TP Customer's trading partner id Trading partner	

### Add Parameters to the Processing Sequence

To add a parameter:

- 1. Click the *Add* button on the toolbar or select the *Parameter* command from the *Insert* menu. You can also add a parameter by selecting a variable in the *Variables* tab and dragging and dropping it over the parameters list.
- 2. Enter or change the parameter name, description, parameter class and default value as required.

### **Delete Parameters from the Processing Sequence**

To delete a parameter:

- 1. Select the parameter to be deleted in the parameters list.
- 2. Click the *Delete* button in the toolbar or select *Delete* from the *Edit* menu.
- 3. Click *Yes* when prompted to confirm the deletion.

## **Cut, Copy or Paste Parameters in the Processing Sequence**

To cut or copy a parameter:

- 1. Select the required parameter in the parameters list.
- 2. Click the *Cut* or *Copy* buttons in the toolbar or select the corresponding command from the *Edit* menu.

To paste a parameter definition that was previously cut or copied, click the *Paste* button in the toolbar or select the *Paste* command from the *Edit* menu.

### **Re-arrange Parameters**

You can re-arrange parameters by moving them up or down the parameter list. To move a parameter up or down the parameter list:

- 1. Select the parameter to be moved
- 2. Click the *Move Up* or *Move Down* button on the toolbar (or select the corresponding command from the *Edit* menu) until the parameter is in the desired position

### **Revise Parameter Definitions**

You can revise the name, description and default value for each parameter. To revise the parameter:

- 1. For the parameter to be revised, click in the column whose value you wish to revise.
- 2. Type or choose the new value for the parameter name, description, parameter class or default value.

### **3.4.6 Edit Processing Sequence Notes**

The *Notes* tab of the *Processing sequence editor* allows you to enter and revise notes associated with the Processing Sequence.

Notes can be used to expand upon the short description to further describe the intended purpose and use of the Processing Sequence. For example you can use the notes to give an overview of the function of the Processing Sequence and when and how to execute it.

### 3.4.7 Save Your Work

You can save your work in the *Processing sequence editor* at any time by clicking the *Save* button on the toolbar or select *Save* or *Save As* from the *File* menu.

If you are creating a new Processing Sequence (or if you chose the *Save As* command), the *Save* window will be shown so that you can give the Processing Sequence a name and a description.

Save			×
	Type name and desc	ription for new processing sequence:	
	Name:	DEM_ORDER_RCV	
	Description:	Order receive processing	
	Status:	Inactive	
	Save	Cancel	

### 3.5 Run a Processing Sequence

You can initiate running a Processing Sequence in several ways.

While you are designing and testing your LANSA Composer solutions, you will usually run a Processing Sequence from within the LANSA Composer client application or from a "shortcut" on the LANSA Composer client computer. Refer to the following sections for more information:

3.5.1 Run a Processing Sequence from the LANSA Composer client

3.5.2 Run a saved Processing Sequence "shortcut" from a client computer

In a production environment, you will usually run a Processing Sequence outside the LANSA Composer client application. This allows you to:

- Initiate running Processing Sequences from your own applications
- Create shortcuts to run Processing Sequences
- Submit Processing Sequence runs as batch jobs on your IBM i server
- Plug-Processing Sequences into your job scheduler of choice.

Depending on your requirements and circumstances, you can choose from one of several means of initiating a Processing Sequence run:

3.5.3 Run a Processing Sequence from the Operations Console

3.5.4 Run a Processing Sequence using the COMPOSER command

- 3.5.5 Run a Processing Sequence using a Run Control File
- 3.5.6 Run a Processing Sequence from a LANSA application
- 3.5.7 Run a Processing Sequence by calling the Web Service function
- 3.5.8 Run a Processing Sequence using the DXP1FN1 function

# **3.5.1 Run a Processing Sequence from the LANSA Composer client**

During development and testing, you will usually run a Processing Sequence from within LANSA Composer. To do this, perform the following steps:

- 1. Expand *Definitions* in the *Navigator* and click *Processing sequences*. Use the filters and the Processing Sequence list to locate and select the Processing Sequence you wish to run.
- 2. Click the *Run* button on the toolbar. The *Run processing sequence* window is displayed.

Note: there are several other ways to initiate running a Processing Sequence, including using the menu option on the *File* menu, the shortcut Ctrl+U, the *Run* button in the *Details* tab, or from within the *Processing sequence editor*.

	Processing sequence	DEM_ORDER_RCV
X	Description	Order receive processing
	Submit to <u>b</u> atch	Run immediately
	Type parameter values fo	r this run:
	Customer's trading partner id	'ATLAS'
	Customer's remote directory	'/neworders'

If there are any parameters defined for the Processing Sequence, the parameters are listed and you can enter values for them. If you enter an alphanumeric literal value you can surround it with quote marks. The quote marks do not become part of the parameter value that the Processing Sequence receives. If you need to embed quote marks in the parameter value, you should surround the value with quote marks and double-up the embedded quote marks.

If the parameter class is other than *General*, you can press F4 or click the

adjacent Prompt (...) button to browse and choose from a list of possible values. For example, if the parameter class is *Trading Partner*, LANSA Composer will display a list of defined Trading Partners. Or if the parameter class is *File path*, LANSA Composer will allow you to browse and choose from files present in the server file system.

When you have entered the parameter values, click the *Run* button.

- If the *Submit to batch* radio button is checked, the Processing Sequence run is submitted to batch using the job description specified in *System Settings*. When the job completes, you can review the result and the log output on the *Run History* tab.
- If you selected the Run immediately button, the Processing Sequence runs immediately and you must wait for it to complete. When the run is complete, the *Processing Sequence Log* window shows the result code and log output for the Processing Sequence run. The amount of detail in the log output depends on the logging level in effect for the Processing Sequence Log window, refer to 3.7.1 Display the Processing Sequence Log from LANSA Composer.

# **3.5.2 Run a saved Processing Sequence** "shortcut" from a client computer

On any computer where the LANSA Composer client is installed, you can also run a saved processing sequence "shortcut" file. You don't have to start the LANSA Composer client application to execute a saved processing sequence "shortcut".

Refer to the following topics for more details:

About Processing Sequence "shortcut" files Save a Processing Sequence "shortcut" file Run a saved Processing Sequence "shortcut" file

## About Processing Sequence "shortcut" files

A Processing Sequence "shortcut" file contains the details necessary to run a Processing Sequence along with options and any parameter values you specify. Amongst the details included in a Processing Sequence "shortcut" file are:

- The Processing Sequence identity
- Any non-default Processing Sequence parameter values you specify when you save the "shortcut"
- The identity of the LANSA Composer server that contains the Processing Sequence

The "shortcut" file is saved as a file with the extension *.dxrun*. LANSA Composer associates this file extension with itself so that when you "*open*" the file in Windows Explorer a LANSA Composer program (dxstart.exe) is started to read the necessary details from the "shortcut" file and execute the Processing Sequence as required, connecting to the LANSA Composer server as required.

The LANSA Composer client must be installed on the computer that executes the Processing Sequence "shortcut" file.

LANSA Composer provides a default folder for holding your Processing Sequence shortcuts, but you can save or copy them to any folder you choose, including onto your desktop or your Windows *Start* menu.

Note that a Processing Sequence "shortcut" file is a "real" file and not a Windows shortcut or *link*. You can however create Windows shortcuts to your Processing Sequence "shortcut" files if you wish.

### Save a Processing Sequence "shortcut" file

You save a Processing Sequence "shortcut" file from the *Run processing sequence* window. For information on opening and using the *Run processing sequence* window refer to:

3.5.1 Run a Processing Sequence from the LANSA Composer client To save a Processing Sequence "shortcut" file from the *Run processing sequence* window, do the following:

- 1. Choose any run options that you wish to apply to the "shortcut" file. For example, click the *Submit* or *Run immediately* buttons according to the default choice you wish to apply to the "shortcut" file.
- 2. Type any non-default parameter values you wish according to the parameter values you wish to be used by the "shortcut" file.
- 3. Click the *Save shortcut* button to display the *Save Processing Sequence Shortcut* window
- 4. Type or choose the desired name and location for the Processing Sequence "shortcut" file
- 5. Click *Save* to save the Processing Sequence "shortcut" file.

Your Processing Sequence "shortcut" will be saved including the options and any non-default parameter values specified in the *Run processing sequence* window. The *Run processing sequence* window is displayed again. If you are finished, you can click Cancel. (Alternatively you can type new values and save a new Processing Sequence "shortcut" for the same Processing Sequence, or you can proceed with the Run request as normal.)

If you saved your Processing Sequence "shortcut" file in the default location suggested by LANSA Composer, it will appear in the *Run processing sequence shortcuts* window in the LANSA Composer client application (you may have to click *Refresh* to see the new shortcut).

### Run a saved Processing Sequence "shortcut" file

You can run a saved processing sequence "shortcut" file on any computer where the LANSA Composer client is installed. You don't have to start the LANSA Composer client application in order to run a saved processing sequence "shortcut".

You can run a saved processing sequence "shortcut" file in two general ways:

1.Because a saved processing sequence "shortcut" file is just a file in the Windows files system (just like a Microsoft Word document file, for example), you can open (run) it in any of the ways that Windows supports.

For example you can click it or double-click it in Windows Explorer, or rightclick and choose *Open* from the pop-up menu. If you have saved the file (or a Windows shortcut to it) to your desktop or your *Start* menu, you can open it from there. Or you can even open it from a command window.

The LANSA Composer client application window does not have to be open in order to *Open* (run) a Processing Sequence "shortcut" file from Windows.

- 2.If the LANSA Composer client application is open and your "shortcut" file is saved in the default location suggested by LANSA Composer, you can run it as follows:
  - a. Expand *Operations* in the *Navigator* and click *Run processing sequence shortcuts*.
  - b. Double-click the desired "shortcut", or select it and click *Run*.

Whichever way you choose to run a saved processing sequence "shortcut" file, the request will usually proceed as follows:

- LANSA Composer displays the *Connect to LANSA Composer Server* window
- Once connected successfully, LANSA Composer displays the *Run processing sequence* window, populated with the processing sequence details, run options and non-default parameter values specified in the shortcut.
- If the *Submit to batch* radio button is checked, the Processing Sequence run is submitted to batch using the job description specified in *System Settings*.

• If you selected the Run immediately button, the Processing Sequence runs immediately and you must wait for it to complete. When the run is complete, the *Processing Sequence Log* window shows the result code and log output for the Processing Sequence run.

Note: the sequence described above may be varied according to options specified in the Processing Sequence "shortcut" file.

### **3.5.3 Run a Processing Sequence from the Operations Console**

LANSA Composer provides a web-browser based interface for operational monitoring and control. One of the functions provided by the Operations Console is the ability to display a list of available and active Processing Sequences and submit a run for selected Processing Sequences. This feature includes the ability to enter values for Processing Sequence parameters, if applicable.

For more information, refer to Operations Console.

# **3.5.4 Run a Processing Sequence using the COMPOSER command**

Note: The COMPOSER command is provided only for IBM i servers. On Windows servers, you must use one of the other documented means of initiating a Processing Sequence run.

LANSA Composer provides the COMPOSER command for invoking selected LANSA Composer functionality on the LANSA Composer server. One of its main functions is to run a Processing Sequence.

On an IBM i server, using the COMPOSER command is the easiest way to integrate your LANSA Composer solutions into your own applications and into your routine system operations.

The COMPOSER command has command parameters that allow you to specify:

- the identifier of the Processing Sequence to run
- values for parameters received by the Processing Sequence

For example, this COMPOSER command can be used to run the supplied processing sequence TUT\_01\_P1 and provide the parameter values for the Processing Sequence:

# <pgmlib>/COMPOSER REQUEST(RUN) pseq(TUT\_01\_P1) parms( (YOURINITIALS 'FS') (YOURMESSAGE 'Hello World') (YOURNAME 'Fred Smith') )

(where **<pgmlib>** is the name of the program library for the LANSA system in which LANSA Composer is installed.)

When connected to an IBM i server, the LANSA Composer client application provides a sample COMPOSER command for running the Processing Sequence. You can copy and adapt the sample command for your own use - in particular, you will often need to change the parameter values. To access the samples, select the *Run commands* tab when a Processing Sequence is selected in the Processing Sequence list.



For more information about using the COMPOSER command, refer to: Appendix D. The COMPOSER Command

### 3.5.5 Run a Processing Sequence using a Run Control File

LANSA Composer provides the DXP1RUN function for running a Processing Sequence using a run control file. The run control file provides the variable information necessary for the run, including the Processing Sequence identifier and any parameter values.

You must supply the Run Control File and furnish its path and name in the arguments to the DXP1RUN function. Depending on the application requirements and the Processing Sequence parameters, your application may need to write or revise a run control file for each run with variable information appropriate for that run.

LANSA Composer provides a sample command line and sample contents of a run control file for running a Processing Sequence in this way. You can copy the samples, complete the missing details and otherwise adapt them for your own use. To access the samples, select the *Run commands* tab when a Processing Sequence is selected in the Processing Sequence list.

📄 Details 📔 Run commands 🔄 Run history 🔚 Version history 🔛 Notes 😣 Audit	
Use X_RUN with a run control file	
Complete this command to run the processing sequence using a run control file	
$\label{eq:xF2PGMLIB} XF2PGMLIB/lansa request(x_run) \ process(dxcontrol) \ function(dxp1run)$	Сору
Sample run control file:	
** LANSA Composer run control file	Сору
* Processing sequence id * - use ONLY one of internal id (DXP1II) or external id (DXP1ID)	
<pre>* DXP1ID=DEM_ORDER_RCV DXP1II=18E7BABDD3E3478796551619F45EEB39</pre>	
* Zero, one or more processing sequence parameter values specified * as NAME=VALUE. Quotes surrounding alphanumeric values are optional	
[Parameters]	
CUSTOMER_TP='ATLAS' CUSTOMER_DIR='/neworders'	

# **Run Control File**

The run control file is a plain text file that is used to specify:

- The Processing Sequence to run
- The values to be used for the Processing Sequence parameters.

Following is an example of a run control file:

```
CUSTOMER_TP='ATLAS'
CUSTOMER_DIR='/neworders'
```

### **Run Control File Notes**

One of the following MUST be provided as the FIRST non-comment line:

- DXP1II=<id> where <id> is the internal identifier of the Processing Sequence to be run.
- DXP1ID=<id> where <id> is the external identifier of the Processing Sequence to be run.

Only the first of these will be processed.

• Parameters may be passed using <name>-<value> pairs. Such parameters must follow a 'section' that begins with the heading [Parameters]. Any parameters found after this heading will be written to the variable pool for the Processing Sequence run. Refer to the example above.

Some relevant points:

- only 1 section heading or name-value pair can be specified on each line.
- the format of the name-value pairs is <name>=<value>
- blank lines are ignored.
- lines with a \* or # as the first non-blank character are comment lines and are ignored.

### The Run Command

To execute the Processing Sequence you must run the function DXP1RUN in process DXCONTROL and provide the path to the run control file in the UDEF= argument.

On Windows servers, you run function DXP1RUN by executing x\_run.exe with the necessary arguments to specify process, function, partition and language, the UDEF= argument, and any other arguments required to specify database and authorisation details. A sample command looks like this:

"C:\PROGRAM FILES\LANSA

```
Composer\X_WIN95\X_LANSA\execute\x_run.exe"
PROC=DXCONTROL FUNC=DXP1RUN UDEF=
<run_control_file> LANG=ENG PART=LIC
USER=PCXUSER PSPW=<password> DBUS=<database
user> PSWD=<database password> DBID=<database
name> DBUT=<database type>
```

LANSA Composer provides a skeletal command on the *Run commands* tab. You can click the *Copy* button to copy the command to the clipboard:



On IBM i servers, you run function DXP1RUN by executing the LANSA command with the process, function, partition and language parameters. You specify the UDEF= argument, with the path to the run control file, in the  $x_runadprm$  parameter. A sample command looks like this:

```
LICPGMLIB/lansa request(x_run) process(dxcontrol)
function(dxp1run) partition(LIC) language(ENG)
x_runadprm('UDEF=<run_control_file>')
```

Again LANSA Composer provides a skeletal command on the *Run commands* tab. You can click the *Copy* button to copy the command to the clipboard:



Refer to LANSA documentation for the details of the arguments to x\_run.exe or the parameters for the LANSA command.

### 3.5.6 Run a Processing Sequence from a LANSA application

Refer to LANSA product documentation for detailed information on application development with LANSA.

The LANSA development environment (as of V12SP1) provides two new builtin functions that allow your LANSA application to quickly and easily execute a LANSA Composer Processing Sequence, either synchronously or asynchronously. The built-in functions are:

- COMPOSER\_USE : specifies the details necessary for your LANSA application to connect to the nominated LANSA Composer server system.
- COMPOSER\_RUN : runs a LANSA Composer Processing Sequence, in the LANSA Composer system identified by the server symbolic name argument, through the LANSA Composer Request Server. It can pass up to five named parameter values to the processing sequence.

Refer to the LANSA Technical Reference for detailed programming instructions for these built-in functions. For information about the LANSA Composer Request Server, refer to:

Appendix F. The LANSA Composer Request Server

# **3.5.7 Run a Processing Sequence by calling the Web Service function**

LANSA Composer implements a SOAP web service that provides a *Run* web service operation for running a Processing Sequence.

For further information, please refer to the following:

Before you begin Services required for the web service support

Overview of the Run web service operation

Accessing the web service from your SOAP client application

Note: This section assumes you already understand the principles and operation of SOAP Web Services and how to invoke them using your chosen software or programming tool or language.

## Before you begin

Note: This section applies ONLY to the LANSA Composer Server on IBM i. If your LANSA Composer Server is installed on Windows, these configuration steps have already been completed by the installation program and you can proceed directly to the next section.

During LANSA Composer IBM i server installation or upgrade, the LANSA Composer installer loads the components necessary to support the LANSA Composer SOAP web service. However, some manual steps must be completed in order to deploy and configure the components before first use.

Note: You only need to perform these steps once after installing or upgrading the LANSA Composer server.

Follow this procedure to deploy and configure the LANSA Composer SOAP web service components:

1. Deploy the LANSAComposerProcessingSequences.soap.jar file

Locate the *LANSAComposerProcessingSequences.soap.jar* file that implements the service. This file can be found in the ./*WebService/SOAP/Deploy* directory of the LANSA Composer data directory tree. For example, on an IBM i server, it can be found on the IFS in a location like this:

/LANSA\_Composer\_licpgmlib/lic/WebService/SOAP/Deploy Copy this file to the *JAR* directory of the JSM instance tree for the JSM installation associated with LANSA Composer, replacing the existing file if necessary. For example, on an IBM i server, the *JAR* directory can be found on the IFS in a location like this:

/LANSA\_licpgmlib/jsm/instance/JAR

2. Deploy the SOAPServerService properties

Locate the *SOAPServerService.properties*. This file can be found in the *./WebService/SOAP/Deploy* directory of the LANSA Composer data directory tree. For example, on an IBM i server, it can be found on the IFS in a location like this:

/LANSA\_Composer\_licpgmlib/lic/WebService/SOAP/Deploy Open the file in a text editor (for example, in *notepad.exe*) and copy the contents. Then locate the *SOAPServerService.properties* already present in the *properties* directory of the JSM instance tree for the JSM installation associated with LANSA Composer. For example, on an IBM i server, the *SOAPServerService.properties* file can be found on the IFS in a location like this:

```
/LANSA_licpgmlib/jsm/instance/properties/
```

Open this file in your text editor, paste/insert the copied lines into the file and save your changes. The altered file contents should look similar to this:

```
#!<studio-project id="20000000-000000" name="lansa">
```

#

service.lansacomposerprocessingsequences=com.lansa.composer.pseq.server.seprovider.lansacomposerprocessingsequences=com.lansa.composer.pseq.server. #

```
# SOAPServerService resources ( Default )
```

#

message.001=Service successfully loaded

```
message.002=Service successfully unloaded
```

```
(more similar lines not shown)
```

# #!</studio-project>

#

3. Create the JSMDirect service entry

The LANSA Composer SOAP web service will be invoked using a URL in a form similar to this:

```
http://SERVER:80/cgi-bin/jsmdirect?COMPOSER_RUN
As you can see, the request will be processed by the CGI program
JSMDIRECT, which is a general purpose CGI interface provided for use
with LANSA Composer (as a part of the included LANSA Integrator).
The COMPOSER_RUN service name specified in the URL instructs
JSMDIRECT exactly which service implementation will process the request.
JSMDIRECT uses service table entries to associate the service name (for example, COMPOSER_RUN) with the executable files that will process the request.
```

The final step of configuration is to create the JSMDirect service table entry for the COMPOSER\_RUN service. There are several ways to do this – you should refer to the supplied LANSA Integrator guide for details. You need to create a service entry containing the following details:

Service	COMPOSER_RUN
Host	*DEFAULT
Process	DXCONTROL
Function	DXP1WS1
Partition	LIC
Language	ENG
Program	
RDMLX	
NSDL Path	

4. Restart the Java Service Manager (JSM)

Finally, you need to restart the Java Service Manager (JSM) in order to make the changes effective.

### Services required for the web service support

In order for the LANSA Composer SOAP web service to respond to in-coming requests, the following services need to be active on your LANSA Composer server:

- The LANSA listener for the LANSA Composer system
- The Web/HTTP server instance associated with the LANSA Composer system
- The Java Service Manager (JSM)

For more information about these services, please refer to the appropriate one of the following headings in Appendix A. Install LANSA Composer:

- For an IBM i server: Subsystems and Jobs
- For a Windows server: Services used by the LANSA Composer Windows Server

### Overview of the Run web service operation

The *Run* web service operation allows your SOAP client application to specify the processing sequence name and any processing sequence parameter names and values necessary to run the Processing Sequence.



The SOAP response will include:

- The result code (OK, ER ...);
- The run number assigned to the run by LANSA Composer (this can be used to identify and view the Processing Sequence Log in LANSA Composer, if necessary);
- The last error message number and text, if the Processing Sequence ended in error;
- The names and values of the Processing Sequence output parameters, if any, unless the Processing Sequence ended in error.



## Accessing the web service from your SOAP client application

In order to successfully invoke the *Run* web service operation, your SOAP client application needs to know and implement the appropriate details in two categories. Refer to the following headings for more information:

Web Service Details

Processing Sequence Details

### Web Service Details

You need to know about the web service in order to execute the correct protocols to invoke it. The most important items you need are:

• The location of the WSDL

LANSA Composer provides the

*LANSAComposerProcessingSequencesService.wsdl* file that describes the service. This file can be found in the *./WebService/SOAP* directory of the LANSA Composer data directory tree. For example, on an IBM i server, it can be found on the IFS in a location like this:

/LANSA\_Composer\_licpgmlib/lic/WebService/SOAP

• The correct endpoint URL for your installation of LANSA Composer The correct endpoint URL necessary to invoke the *Run* web service operation will depend on the specifics of the environment in which LANSA Composer server is installed.

The provided WSDL specifies the endpoint URL generically as follows:

http://SERVER:80/cgi-bin/jsmdirect?COMPOSER\_RUN

You need to revise this for your system by:

- Replacing *SERVER* with the DNS name of your server system on which LANSA Composer is installed
- Replacing the port number (80) with the HTTP listener port number specified during the LANSA Composer server installation.
- If you are running LANSA Composer on a Windows server, you need to replace *jsmdirect* with *jsmdirect.exe*.

For example, if LANSA Composer is installed on a Windows server named MYSERVER and the HTTP listener port in use is 8082, then your endpoint URL might look like this:

http://MYSERVER:8082/cgi-bin/jsmdirect.exe?COMPOSER\_RUN

#### **Processing Sequence Details**

You need to know about the Processing Sequence that you wish to run:

• The Processing Sequence name

You must specify the Processing Sequence name in the *processingSequenceName* parameter for the *Run* web service operation. In the illustration below, the Processing Sequence name specified is EXAMPLE\_AATEST2.

• The Processing Sequence input parameter names and the values you need to supply for them.

You must specify one *parameterName* and *parameterValue* pair for each Processing Sequence parameter for which you wish to specify a value. In the illustration below, one Processing Sequence parameter value is specified using the name DIRECTORY and the value shown.

```
<scapenv:Envelope xmlns:scapenv="http://schemas.xmlscap.org/scap/envelope/" xmlns:scap="ht
   <soapenv:Header/>
   <soapenv:Body>
     <soap:Run>
         <soap:processingSequenceName>EXAMPLE_AATEST2</soap:processingSequenceName>
         <soap:processingSequenceInputs>
            <soap:processingSequenceParameters>
               < !-- Zero or more repetitions: -->
               <soap:item>
                 <soap:parameterName>DIRECTORY</soap:parameterName>
                 <soap:parameterValue>/LANSA Composer licpgmlib/lic/Tutorial/</soap:param</pre>
              </soap:item>
           </soap:processingSequenceParameters>
         </soap:processingSequenceInputs>
     </soap:Run>
  </soapenv:Body>
</soapenv:Envelope>
```

### 3.5.8 Run a Processing Sequence using the DXP1FN1 function

IMPORTANT NOTE:

This section describes a technique for running a Processing Sequence from a LANSA application that is no longer recommended for new solutions. It is only useful when LANSA Composer is installed in the same LANSA run-time system as the LANSA application. This mode of installation is no longer possible for new installations and is not recommended. For new solutions, please refer to 3.5.6 Run a Processing Sequence from a LANSA application instead.

NOTE: This section assumes sound knowledge of the applicable LANSA development techniques. Refer to LANSA product documentation for detailed information on application development with LANSA.

LANSA Composer provides the DXP1FN1 function for running a Processing Sequence from a LANSA application executing in the same LANSA system and partition as LANSA Composer.

Your LANSA application can call DXP1FN1 using the CALL RDML command with PROCESS(\*DIRECT):

- Function DXP1FN1 must execute on the LANSA Composer server. Your application may, however, execute it via the LANSA SuperServer CALL\_SERVER\_FUNCTION built-in function.
- Function DXP1FN1 executes the Processing Sequence synchronously. If you wish the Processing Sequence to run in a batch job, then you must submit the code unit in your application that calls function DXP1FN1.
- Your application can pass the necessary variable information (the processing sequence identifier and any parameter values) using a combination of the LANSA exchange list and by passing a working list.
- Your application can receive the result of the processing sequence run via the exchange list.

NOTE: The code unit in your application that calls function DXP1FN1 must be RDMLX enabled.

Refer to the following topics for more details:

Exchange the Processing Sequence identifier

Populate and pass the Parameters working list

Receive the result of the Processing Sequence run

Example function to run a Processing Sequence

For general considerations about integrating your LANSA Composer solution with your LANSA applications refer to:

Appendix E. Using LANSA Composer with LANSA Applications

## **Exchange the Processing Sequence identifier**

Before calling function DXP1FN1, your application must place the Processing Sequence identifier on the exchange list using the EXCHANGE RDML command:

- you must use the *internal* identifier shown on the *Audit* command tab for the Processing Sequence
- you must assign the value to field DXP1II and place it on the exchange list immediately before the function call
- field DXP1II must be defined as alphanumeric with length 32.

(If you have imported the LANSA Composer development package into your Visual LANSA development environment, the definition of field DXP1II will already be present in your LANSA repository.)

NOTE: Function DXP1FN1 can receive the values of other fields with names beginning with 'DX' from the exchange list. Your application does not need to (and should not) exchange these other fields. To avoid interfering with the correct behavior of function DXP1FN1, you should NEVER exchange any fields with names beginning with 'DX' before the call, other than the fields explicitly described here.

For an example of exchanging the Processing Sequence identifier before calling function DXP1FN1 refer to:

Example function to run a Processing Sequence

### Populate and pass the Parameters working list

Function DXP1FN1 expects to receive a working list containing zero, one or more (up to 100) entries defining processing sequence parameter values that are to be used by the processing sequence run. You must pass this working list to function DXP1FN1 - if your processing sequence does not use any parameters, you must still pass an empty working list.

You must define the working list in your program using exactly the same names and characteristics as the working list used by function DXP1FN1. It should be defined as follows:

\* \_\_\_\_\_

\* working list for processing sequence parameters

\* \_\_\_\_\_

def\_list name(#dxparms) fields(#dxapnm #dxpvsq #dxpvvl)
type(\*working) entrys(100)

If you have imported the LANSA Composer development package into your Visual LANSA development environment, the definition of the working list fields will already be present in your LANSA repository. Otherwise these fields must be defined with these essential characteristics:

Name	Description	Туре
DXAPNM	Parameter name	A(20)
DXPVSQ	Sequence number	P(5, 0)
DXPVVL	Parameter value	A(256)

Your application should set the values of the working list fields and add an entry to the working list for each parameter for which you wish to provide a value.

NOTE: your application should set the value of the DXPVSQ (sequence number) field to 1 (one) for every working list entry. The Processing Sequence will not receive the parameter value if you
assign any other value to the DXPVSQ field.

For an example of defining and populating the Parameters working list before calling function DXP1FN1 refer to:

Example function to run a Processing Sequence

### **Receive the result of the Processing Sequence run**

When function DXP1FN1 completes, it places several variables on the LANSA exchange list that identify the processing sequence run and provide the result. Your application may refer to these fields immediately after the CALL to determine the result.

If you have imported the LANSA Composer development package into your Visual LANSA development environment, the definition of the necessary fields will already be present in your LANSA repository. Otherwise, if you wish to refer to them, these fields must be defined with these essential characteristics:

Name	Description	Туре
DXP1RN	Run number	P(15, 0)
DXPVRN	Run number (character format as displayed in LANSA Composer)	A(15)
DXEXRC	Execution result code	A(2)
DXEXSV	Execution result severity	P(3, 0)

For an example of receiving and testing the result of the Processing Sequence run after calling function DXP1FN1 refer to:

Example function to run a Processing Sequence

### **Example function to run a Processing Sequence**

This example function runs the supplied TUT\_01\_P1 processing sequence. It demonstrates all the key aspects of running a Processing Sequence from a LANSA application by calling function DXP1FN1:, including:

Exchange the Processing Sequence identifier Populate and pass the Parameters working list Receive the result of the Processing Sequence run

\* \_\_\_\_\_

\* This function runs the supplied TUT\_01\_P1 processing

\* sequence, supplying parameter values as required

\* \_\_\_\_\_

function options(\*direct)

\* \_\_\_\_\_

\* working list for processing sequence parameters

\* \_\_\_\_\_

def\_list name(#dxparms) fields(#dxapnm #dxpvsq #dxpvvl) type(\*working)
entrys(100)

\* \_\_\_\_\_

\* set and exchange the processing sequence identifier
\* (uses the internal identifier as shown on the Audit tab)

#dxp1ii := '2D123D3124F84042AFB1E931838DA90E'
exchange fields(#dxp1ii)

\* \_\_\_\_\_

\* provide named parameter values for the run

\* \_\_\_\_\_

#dxpvsq := 1

#dxapnm := 'YOURINITIALS'

```
#dxpvvl := 'LA'
add_entry to_list(#dxparms)
```

#dxapnm := 'YOURMESSAGE'
#dxpvvl := 'This run was initiated by my LANSA application'
add\_entry to\_list(#dxparms)

#dxapnm := 'YOURNAME'
#dxpvvl := 'LANSA Application'
add\_entry to\_list(#dxparms)

\* \_\_\_\_\_

\* run the processing sequence

\* \_\_\_\_\_

call process(\*direct) function(DXP1FN1) pass\_lst(#dxparms)

\* \_\_\_\_\_ \* test the result \* \_\_\_\_\_

if (#dxexrc = 'OK')

message msgtxt('Processing sequence run was successful')

else

message msgtxt('Processing sequence run ended abnormally')

endif

\* \_\_\_\_\_

\* that's all folks!

\* \_\_\_\_\_

return

## 3.6 Restart a Processing Sequence Run

When a Processing Sequence run ends in error (including when its execution is suspended explicitly using the SUSPEND Processing Sequence directive), it may be possible to restart it from the point of failure.

For example, if an FTP\_INBOUND Activity in a Processing Sequence fails, perhaps because the remote FTP location was unavailable, it is usually possible to restart the Processing Sequence at a later time, when the communication error has been resolved.

When a failed Processing Sequence restarts:

- It restarts from the point of failure, even if the point of failure was inside one or more Processing Sequence "blocks", such as a LOOP or WHILE block, or in a conditional block such as IF or CASE.
- Any loop indices and all Processing Sequence variables retain the values they had at the point of failure.
- The Processing Sequence executes the same version of the Processing Sequence definition with the Processing Sequence run was started, even if that is not the current definition.

Note: If the Processing Sequence execution was suspended explicitly using the SUSPEND Processing Sequence directive, the Processing Sequence restarts immediately *after* the SUSPEND directive.

#### Conditions for restarting a Processing Sequence run

In order for a Processing Sequence run to be eligible to be restarted, the following conditions must be true:

- The Processing Sequence definition must specify that is restartable.
- The definition of the Activity or Transformation Map that ended in error must specify that it is restartable.
- There must have been no active aXes terminal sessions active at the point of failure.
- The version of the Processing Sequence definition that ended in error must still be available.
- The run history for the affected Processing Sequence run that ended in error must still be available.

• The error arose through a Processing Sequence error status being raised. Processing Sequence that end in error as a result of abnormal program termination of a program used by an Activity, Transformation Map or the Processing Sequence engine cannot be restarted.

Note: Processing Sequence run history and version history can be purged using the Database Housekeeping function. If the necessary information has been purged, it will not be possible to restart the affected Processing Sequence run.

### **3.6.1 Restart a Processing Sequence Run from LANSA Composer**

You can restart an eligible Processing Sequence run from within the LANSA Composer client application. To do this, perform the following steps:

- 1. Expand *Definitions* in the *Navigator* and click *Processing sequences*. Use the filters and the Processing Sequence list to locate and select the desired Processing Sequence.
- 2. Click the *Run History* tab. A list of recorded prior runs of the Processing Sequence will be displayed.
- 3. Select the run you wish to restart and click the *Restart* button (if the *Restart* button is not enabled, then the run is not eligible to be restarted). The *Restart processing sequence* window is displayed.

🕨 Restart I	Processing Sequence		
1	Processing sequence Description	DEM_ORDER_RCV Order receive processing	
	O Submit to <u>b</u> atch	💌 Run immediately	]
	Run number	00000000000071	]
	Start date / time	2/12/2008 6:36:18 AM	]
	End date / time	2/12/2008 6:36:21 AM	]
	Execution result code	Error	]
	Re <u>s</u> tart	ancel	

- 4. Click the Restart button.
  - If the *Submit to batch* radio button is checked, the Processing Sequence run is submitted to batch using the job description specified in *System Settings*. When the job completes, you can review the result and the log output on the *Run History* tab.
  - If you selected the Run immediately button, the Processing Sequence runs immediately and you must wait for it to complete. When the run is complete, the *Processing Sequence Log* window shows the result code and log output for the Processing Sequence run. The amount of detail in the

log output depends on the logging level set in the LANSA Composer system settings.

# **3.6.2 Restart a Processing Sequence Run from the Operations Console**

LANSA Composer provides a web-browser based interface for operational monitoring and control. One of the functions provided by the Operations Console is the ability to display a list of active and completed Processing Sequences runs and restart a failed run, if eligible for restarting.

For more information, refer to Operations Console.

### **3.7 Review the Processing Sequence Log**

One of the most powerful features of LANSA Composer is the logging for completed Processing Sequence runs. The Processing Sequence Logs can be displayed and printed in a variety of ways and with varying levels of detail. But with default settings LANSA Composer will usually provide a view and a level of detail that is appropriate to your interest and to the status of the run (that is, whether it completed successfully or not).

While you are designing and testing your LANSA Composer solutions, you will usually display the Processing Sequence Log from within the LANSA Composer client application. Refer to the following heading for more information:

### 3.7.1 Display the Processing Sequence Log from LANSA Composer

In a production environment, you may monitor your Processing Sequence runs from outside the LANSA Composer client application. Refer to the following heading for more information:

3.7.2 Display the Processing Sequence Log from the Operations Console

# **3.7.1 Display the Processing Sequence Log from LANSA Composer**

When you *run* or *restart* a Processing Sequence run from within the LANSA Composer client application using the *Immediate* option, LANSA Composer will automatically display the *Processing Sequence Log* window when the run completes.

In addition you can choose to display the *Processing Sequence Log* window for previously completed Processing Sequence runs in a number of ways, including, for example, from the *Run History* command tab.



The *Processing Sequence Log* window lets you view or print the Processing Sequence log in a variety of ways. Some of the things you can do from this window are described under the following headings:

Print the Processing Sequence Log End an active Processing Sequence run Restart a failed Processing Sequence run Change the level of logging detail shown

#### Change your view of the log information

### **Print the Processing Sequence Log**

You can click the *Print* button or choose *Print* from the *File* menu to print the Processing Sequence Log.

When you print the Processing Sequence Log from the *Processing Sequence Log* window, LANSA Composer will print the log "as displayed" – that is, log messages that are collapsed or expanded will be printed in the same way. So to print the log exactly as you want it, first expand and/or collapse items as required.

Note: this differs from when the Processing Sequence Log is printed directly from the *Run History* command tab. In the latter case, LANSA Composer decides how much of the log is collapsed or expanded in order to give you the most appropriate view.

## End an active Processing Sequence run

When displaying the log for an "active" processing sequence run, you can click the *End* button or choose *End* from the *File* menu to request "controlled end" of the processing sequence run.

Note: The processing sequence controller does NOT automatically respect such requests. It is up to the solution designer to build in such support, where appropriate, by reference to the \*SHUTDOWN builtin variable. It may not always be necessary, but it is advised for processing sequences that are intended to be long-running processes. This would usually include, for example, any processing sequences that use the new WATCH\_MSGQ, WATCH\_DIRECTORY or WATCH\_DTAQ activities, or any that implement similar "monitor" style processing using LOOPs or other constructs.

### **Restart a failed Processing Sequence run**

If the Processing Sequence run ended in error and can be restarted, you can click the *Restart* button or choose *Restart* from the *File* menu to restart the run from the point of failure.

# Change the level of logging detail shown

For Processing Sequence runs that have ended successfully, LANSA Composer normally shows by default a summary view of the log. Conversely, if the run

ended in error, LANSA Composer will show by default a detailed view of the log that normally contains more information that can help to diagnose the cause of the failure.

In any event (but subject to System Settings relating to the logging level), you can switch between the detailed and summary views by clicking the *Show Summary* or *Show Details* buttons or using the equivalent options on the *View* menu.

In addition, and whether in detail or summary view, LANSA Composer may automatically collapse some log entries related to prior successfully completed steps. You can click the '+' or '-' buttons beside the log entries to expand log entries you are interested in or collapse those that you are not interested in.

You can expand all entries in one go by clicking the *Expand All* button or using the corresponding *View* menu item. Note that expanding all entries may take a long time for large Processing Sequence logs.

# Change your view of the log information

You can change your view of the Processing Sequence log in other ways.

In particular, you can show or hide log message details either in a popup window or using a second pane beside the log display. Use the appropriate *View* menu items to do this. The message details pane or popup windows shows details for the currently selected log entry.

# **3.7.2 Display the Processing Sequence Log from the Operations Console**

LANSA Composer provides a web-browser based interface for operational monitoring and control. One of the functions provided by the Operations Console is the ability to display a list of active and completed Processing Sequences runs and to display the Processing Sequence Log for each.

For more information, refer to Operations Console .

# 4. Transaction Document Processing

LANSA Composer's transport, transformation and process orchestration features provide an infinitely flexible and adaptable framework that can be used to solve almost any type of business process integration challenge.

Many typical business process integration requirements follow a similar and familiar pattern. This pattern is characterized by:

- Exchanging transaction documents such as EDI, XML or CSV with trading partners
- Mapping transaction information between the transaction document and an application database
- Executing application-specific functionality to process the transactions
- Exchanging transaction acknowledgements with the originating trading partner

Because this pattern is so typical, LANSA Composer provides a ready-made framework that you can readily adapt and extend to encompass the exchanges and document types that are specific to your organization. As a part of this framework solution, LANSA Composer provides a transaction document registration database and the LANSA Composer Document Manager that can be used to monitor and manage your inbound and outbound document flows.

To better understand the place of the transaction document processing framework and features in LANSA Composer, you might think of it this way:

- LANSA Composer's transport, transformation and process orchestration tools can be used to solve virtually any type of business process integration problem.
- LANSA Composer's transaction document processing frameworks provide ready-made and adaptable solutions, using those tools, for the most common business process integration scenario.
- Using the LANSA Composer toolset, you can adapt the supplied framework to meet your exact circumstances and requirements.

Because EDI is a particularly specialized discipline, the transaction document processing framework and the LANSA Composer Document Manager contain specific features applicable to EDI document exchange. If you do not exchange EDI documents, you can simply disregard those parts of the provided solution.

Refer to the following for more information:

- 4.1 Transaction Document Processing Framework
- 4.2 LANSA Composer Document Manager

## **4.1 Transaction Document Processing Framework**

The LANSA Composer transaction document processing framework provides a ready-made and adaptable solution for the most common business process integration scenarios.

The following sections provide detailed information about the transaction document processing framework:

- 4.1.1 Overview of the Transaction Document Processing Framework
- 4.1.2 Getting Started with the Transaction Document Processing Framework
- 4.1.3 Document Types and Document Standards
- 4.1.4 Trading Partner support for Transaction Document Processing
- 4.1.5 Activities for Transaction Document Processing
- 4.1.6 Processing Sequences for Transaction Document Processing
- 4.1.7 Transformation Maps for Transaction Document Processing
- 4.1.8 Pre-built EDI X12 solution components
- 4.1.9 Application program interfaces (APIs)

# **4.1.1 Overview of the Transaction Document Processing** Framework

LANSA Composer provides a ready-made framework solution for transaction document processing. The framework can very quickly be modified and extended to:

- Handle specific types of transaction documents
- Take account of processing requirements specific to the customer's trading and processing environment.

This framework encompasses supplied LANSA Composer definitions and features working together to provide a near ready-to-use transaction document processing solution.

# Document types and standards

LANSA Composer provides support for *Document types* and *Document standards* definitions that describe certain characteristics of transaction documents such that inbound and outbound document processing can recognize them, validate them and invoke the appropriate processing sequence to further process them.

A core set of *Document types* and *Document standards* definitions are supplied with LANSA Composer for the document types most commonly used, but you can extend these with your own definitions according to your organizations requirements and trading relationships.

For more information on creating or modifying *Document types* and *Document standards* definitions, please refer to:

4.1.3 Document Types and Document Standards

# **Trading Partners**

The use of Trading Partner definitions and features is central to the operation of the supplied transaction document processing framework. Trading Partner definitions provide links to directories, configurations and transformation maps used to process transaction documents. In addition, the Trading Partner Data Interchange attributes provide additional values used in transaction document processing. For more information refer to: 4.1.4 Trading Partner support for Transaction Document Processing.

# Activities

Of course, transaction document processing solutions can and do make use of

any or all of the supplied LANSA Composer Activities.

However, there is a subset of Activities supplied with LANSA Composer that are very specifically intended for and associated with the transaction document processing framework. These are listed in:

4.1.5 Activities for Transaction Document Processing

#### **Processing Sequences**

A core part of the transaction document processing framework is the set of processing sequences that have been pre-built and supplied with LANSA Composer for this purpose. These are listed and further described in:

4.1.6 Processing Sequences for Transaction Document Processing

## **Transformation Maps**

Transformation maps are almost always an integral part of a transaction document processing solution. The pre-built processing sequences that are central to the transaction document processing framework assume a particular pattern for the transformation maps that it invokes and LANSA Composer supplies "model" transformation maps that match this pattern. For more information refer to:

4.1.7 Transformation Maps for Transaction Document Processing

### **Pre-built EDI X12 solution components**

LANSA Composer is supplied with a limited number of pre-built transformation maps, staging database tables and export and import processors and document viewers for selected EDI X12 transactions.

These pre-built components may serve as a pro-forma solution for those transactions, but more especially as examples of a completed solution upon which you might base a solution for other types of transaction documents used by your organization. For more information, refer to:

4.1.8 Pre-built EDI X12 solution components

# Application program interfaces (APIs)

LANSA Composer allows you to extend the transaction document processing framework in several ways by providing application program interfaces (APIs) in the forms of ancestor components for your own LANSA components and by providing callable LANSA interfaces. These interfaces allow you to provide extended support for specific transaction types and to better integrate the solution into your own applications. For more information, refer to: 4.1.9 Application program interfaces (APIs)

# **4.1.2 Getting Started with the Transaction Document Processing Framework**

This section contains some tips and suggestions to guide you in getting started with the LANSA Composer transaction document processing features.

To begin with, you may wish to Run a simple demonstration using the framework as supplied.

Then you can refer to the following for further Getting Started tips:

Determine the scope and subject of a pilot implementation

Planning your implementation

Understand the major implementation steps

If you have an existing solution that you wish to adapt to use the transaction document register, you can refer to:

Adapting your own Solution to Use the Transaction Document Register

### Run a simple demonstration

You can follow the steps of this simple demonstration to simulate running an incoming sales order XML file through LANSA Composer's transaction document processing framework and update the tutorial sales order database with the details contained in the transaction document.

This demonstration assumes that you have an understanding of the basics of how to navigate and use LANSA Composer's main features. You may wish to complete the LANSA Composer tutorials first.

In addition to the Trading Partner definition that you will create as a part of the demonstration and the supplied Activities used by the processing sequences, the following supplied LANSA Composer definitions are used. In order for the demonstration to succeed, these definitions should be in their supplied form and, where applicable, they must be *Active* status:

IDOC_ARCHIVE	Trading Partner directory types
IDOC_PROCESS	defined in Code Maintenance.
IDOC_RECEIVE	
XML	Document type definition
XML/TUTORDERS	Document standard definition
ITUT_ORDERS	Transformation map definition
TXDOC_DATABASE (*)	Database configuration
TXDOC_INBOUND	Processing sequences
TXDOC_IXML010	
TXDOC RCVTXDOC RCV DEMO	

(\*) Note that this demonstration assumes that the TXDOC \_DATABASE configuration has already been modified to suit your installation and that it provides access to the database schema or library containing LANSA Composer's tutorial orders database tables. Refer to Modify the Supplied TXDOC\_DATABASE Database Configuration for more information. WARNING: This demonstration will allocate a transaction document envelope number from LANSA Composer's built-in series and will update the transaction document register with details of the demonstration transaction being processed. Depending on your organisation's procedures and audit requirements, you may not wish to execute this demonstration in a production LANSA Composer environment.

When you are ready, follow these steps to complete the demonstration:

1. Create a new trading partner named iii\_PARTNER1, where iii are your initials. Make sure you set the status to *Active*. In the *Receive processing sequence* field, type or select the supplied processing sequence TXDOC\_RCV\_DEMO.

🍓 Trading partner		
Name	III_PARTNER1	Save
Description	Demo trading partner 1	Refresh
Status	Active	
Contact		
Address 1		
Address 2		
Address 3		
Email	[]	
Email 2		
Email 3		
Phone number		
Fax number		
Receive processing sequence	TXDOC_RCV_DEMO	
Send processing sequence		

(The TXDOC\_RCV\_DEMO processing sequence that you have associated with this trading partner will "receive" transaction document files by copying the supplied tutorial sales order XML file (TUTorder.xml) from the tutorial files directory to the inbound receive directory for this trading partner.)

When prompted, let LANSA Composer create the linked directories for the Trading Partner. (Make sure the *Create directories if they do not exist* checkbox is checked.)

Trading Partner III\_PARTNER1 - Add Linked Directories

LANSA Composer can link directories of the types shown below for this trading partner. If necessary, LANSA Composer can also create the specified directories.

- 🗆 ×

Check the directory types that you want LANSA Composer to link to this trading partner. Type the directory path to use for each selected directory type.

			Directory Type	Directory Path	-			
1	₽	IDOC_ARCHIVE	Inbound transaction doc	/LANSA_Composer_licpgmlib/lic/III_PARTNER1/IDOC_ARCHIVE/				
2		IDOC_INERROR	Inbound transaction doc	/LANSA_Composer_licpgmlib/lic/III_PARTNER1/IDOC_INERROR/				
3	P	IDOC_PROCESS	Inbound transaction doc	/LANSA_Composer_licpgmlib/lic/III_PARTNER1/IDOC_PROCESS/				
4		IDOC_RECEIVE	IDOC_RECEIVE Inbound transaction doc /LANSA_Composer_licpgmlib/lic/III_PARTNER1/IDOC_RECEIVE/					
5		ODOC_ARCHIVE	DOC_ARCHIVE Outbound transaction dv /LANSA_Composer_licpgmlib/lic/III_PARTNER1/0D0C_ARCHIVE/					
6	P	ODOC_INERROR Outbound transaction dr /LANSA_Composer_licpgmlib/lic/III_PARTNER1/0D0C_INERROR/						
7		ODOC_PREPARE Outbound transaction dr /LANSA_Composer_licpgmlib/lic/III_PARTNER1/0D0C_PREPARE/						
8		DODC_SEND Outbound transaction di /LANSA_Composer_licpgmlib/lic/III_PARTNER1/0D0C_SEND/						
✓ Create directories if they do not exist								

2. On the *Linked Maps* tab for the Trading Partner, add a link for the supplied *ITUT\_ORDERS* transformation map.

🍓 Trading partner : Linked maps (III_PARTNER1-Demo trading partner 1)												
📅 Outbound numbering   🛄 Notes   🐼 Audit   Properties   🍕 Data interchange   🗿 Linked directories   🚱 Linked maps   🚰 Linked configurations												
Γ		Map type	Seq	Ack	P/T	Map ID	Description		Map direction	Map standard	Trans type	E Save
	1	specified typ	5	No	Production	ITUT_ORDERS	Inbound XML to tutorial orde	er data	Inbound	<b>4L TUTORDERS</b>	ORDERS	
												Refresh
												Add
												<u>R</u> emove

(This map reads a tutorial sales order XML file (as used in the LANSA Composer tutorial exercises) and writes the contents to LANSA Composer's tutorial orders database.)

3. Run the supplied inbound Processing Sequence, TXDOC\_INBOUND, specifying your trading partner id in the first parameter:

Nun Proce	essing Sequence	
1	Processing sequence Description	TXDOC_INBOUND Process inbound transaction documents
	Submit to <u>b</u> atch	Run immediately
	Type parameter values for	this run:
	Receive for trading partner id	'III_PARTNER1'
	Spawn processes for receive	'N'
Close	R <u>un</u> C	ancel <u>S</u> ave shortcut

- 4. When the processing sequence has completed normally, amongst other things, you should be able to observe:
  - That the inbound transaction document has been renamed and moved to the IDOC\_ARCHIVE directory for the Trading Partner.
  - That records have been added to LANSA Composer's tutorial database tables TUTORDH and TUTORDL containing the contents of the incoming sales order from the TUTorder.xml file.
- 5. Start the *LANSA Composer Document Manager* application (you can use the option from the *Tools* menu in LANSA Composer or the installed program shortcut). Select the *Documents* option from the *Navigator* and then click the *Search* button in the filter pane to locate your transaction document. Click the *History* tab to see the processing history for the transaction document.

	🖸 LANSA Composer Document Manager										
Eil	<u>File View Go</u> <u>H</u> elp										
-		🕜 Start Here	🖧 P	rint							
D	Documents										
											x
		Document Id	Trading Partner	Direction	Split	Exchange Date/Time	Excha	nge Status	Docu	Document Location	ı
	*	⊞ 001	III_PARTNER1	Inbound	Stan	2010-07-06 16:16:31.755	OK - S	Successful	XML	/LANSA_Composer	_licpgmlib/lic/II
											x
		Document :	History (001-III_	PARTNER	1)						
	ſ	View Oth	istory	tion Errors							
				don En ora	_			-			
		1 Registered	transaction docum	ent	Pr	ocessing sequence run numb	er 🧭	000000000	000004		🔄 Refresh
*		2 Processing	begun for transact	tion documer	nt Ex	change log sequence		00000000	,		
		3 Discovered	document type		-			0010.07.0			
		4 Discovered	content type		E)	(change log date/time		2010-07-00	5 16:24:18	5.245	
		5 Found tran	sformation map for	document	E	change result code		OK			
		6 Transforma	ation successful for	document	E)	change status		Successful	v complete	ed processing docur	
		7 Successful	y completed proces	ssing docume	ent						
	Document type XML										
	Exchange content type ORDERS										
					E	change file location		A ANSA CO	omposer li	comlib/lic/III PAR	
F		· · · · · · · · · · · · · · · · · · ·	******	*****		Messa	ages 👘	Ready 0	OMPOSER	01 ENG LICPO	
-										,,	

Congratulations ... you have successfully received your first transaction document through LANSA Composer's transaction document processing framework!

## Determine the scope and subject of a pilot implementation

You should plan for a pilot implementation using a limited but representative selection of trading partner(s) and transaction document types.

This will enable you to more fully understand LANSA Composer's features and the capabilities of the transaction document processing framework with lower risk and provide the opportunity to adapt your implementation to best suit your circumstances.

# **Planning your implementation**

Amongst the tasks you need to perform when planning your implementation is to gather information about the following:

#### **1.** Identify document types and standards applying to your implementation

- Identify the document types (for example, EDI, XML, CSV) that your implementation will use and how they will be represented in LANSA Composer.
- Identify the particular standards and/or versions that apply to those document types and how they will be represented in LANSA Composer.
- Locate and acquire the references that your implementation will use to effect mapping (and validation, if applicable) of the transaction document types and standards you have implemented. For XML documents, for example, you will need access to the relevant document DTD or schema. For text or CSV documents, you will need access to a reference that completely describes the document format. For EDI documents, you will need access to the SEF/LSEF file that describes the standard and you will need to connect that to the applicable LANSA Composer document standard for use by EDI-specific parts of the framework such as the DISCOVER\_EDI activity.

#### 2. Identify your trading partners and trading agreements

- Identify the trading partners with whom you will exchange transaction documents.
- Understand any trading agreements that may govern the transaction document exchange with those trading partners.
- If applicable, categorize your trading partners and agreements such that you can identify the processes that will be common. Assuming that you interact with at least some groups of your trading partners in the same ways, your objective should be to build common processes (or adapt the supplied processes) such that they can serve more than one trading partner relationship. LANSA Composer contains specific features to support this.
- Gather information for each trading partner that will enable you to implement the agreed document exchange. Of course this includes the agreed document types and standards, but also you need to gather the specific details that will enable you to implement the agreed transport protocols, such as FTP address and credentials.

### 3. Understand your application(s) with which the solution will integrate

- Identify your existing application(s) that will be involved in the transaction document processing.
- Understand the relevant parts of the application's database and any rules that govern it.
- Determine the division of responsibilities between the application and your transaction document processing solution. For example, define the extent to which each of them is responsible for cleaning and validating any in-coming transaction data in order to meet the application's processing requirements.
- Identify integration touch-points. Clearly this must include shared database access, but also any program-level integration. For example, the inbound process may need to call an application program or start an application process on receipt of an in-coming transaction. Or your application may need to start the outbound process in response to application events that indicate that an outbound transaction document is to be sent.

#### 4. Determine the most appropriate approach to mapping transaction data

- Decide whether the transformation maps that you will create for your transaction document processing requirements should directly address existing application database tables or instead should use an intermediate "staging" database:
  - Having your transformation maps directly address the application database tables is usually the quickest and simplest solution to implement. However, this may not be appropriate if this results in the solution effectively bypassing data cleansing and integrity provisions of your existing applications.
  - Using an intermediate "staging" database helps to insulate your application from the interfaces that the transaction document processing solution opens to the outside world and affords you the opportunity to implement comprehensive data cleansing and integrity measures. However, this approach is somewhat more complex and costly to implement and may require you to perform additional coding to provide import and/or export processors that support this approach (though it will often be possible to re-use or adapt existing programs for this purpose).

#### Integration with operational procedures, if necessary

• Adapt your operational procedures to accommodate your transaction document processes. Depending on your trading partner agreements, there are a number of ways in which transaction document exchange with your

trading partners may be initiated. Some possibilities are that you may need to add periodic processes to your operational task scheduler or batch processing streams to poll for available inbound transaction documents.

• Allocate responsibility for monitoring and acting upon exceptions that occur in your transaction document flows. You may wish to take advantage of LANSA Composer's in-built event notification features to have selected groups of users automatically notified of exceptions that occur.

# Understand the major implementation steps

In LANSA Composer, as by any other means, there may be a wide variety of steps necessary for a successful transaction document processing solution. This section summarizes some of the major steps and considerations:

# 1. Review and revise LANSA Composer system configuration

Review and revise your LANSA Composer system settings if necessary. Make sure that you have a fully-functioning LANSA Composer system before you begin to implement a transaction document processing solution.

# 2. Review and revise document types and standards

Review the existing document types and document standards, creating new ones or revising the existing definitions to suit your implementation.

# **Remember:**

- " The DISCOVER\_DOC activity uses the file extension associated with a Document type to determine the document type associated with a transaction document.
- " The TXDOC\_INBOUND and TXDOC\_OUTBOUND processes use the inbound and outbound processing sequences associated with the document type to perform further processing of the document.

# **Remember:**

- " For EDI documents, the DISCOVER\_EDI activity uses the match agency and match version values associated with each document standard to identify the LANSA Composer document standard definition that applies to the EDI transaction document
- " For EDI documents, the DISCOVER\_EDI activity uses the SEF file name specified for the document standard to perform EDI validation, if required.

# 3. Create inbound and/or outbound transformation maps

Having identified the document types, standards and transaction types that your implementation is required to support, you will need to create transformation maps that can map between the transaction data and your staging database or your application database, according to the approach you have chosen.

Remember that in the standard inbound and outbound process, the transformation maps are assumed to have a basic structure (to the extent that it

affects the parameters they receive). Refer to Model Transformation Maps for Transaction Document Processing for more information.

If you exchange the same transaction types with more than one trading partner, you should start with the objective of re-using the same transformation maps for each trading partner to whom it applies. There may be cases however where this is not possible or appropriate. Through its trading partner Linked Maps feature, LANSA Composer supports either possibility.

# **Remember:**

- " LANSA Composer uses the map direction, standard and transaction type specified in the data interchange attributes for a transformation map (along with information specified when the map is linked to a trading partner) to identify the transformation map that applies to a transaction document in the FIND\_TPMAP activity.
- " The inbound process and outbound process reference the import and export processors associated with a transformation map (if applicable) by referencing the corresponding Transformation Map built-in variable and/or by means of the DISCOVER\_MAP activity

# 4. Create trading partner definitions and transport configurations

You will need to create trading partner definitions for each trading partner with whom you exchange transaction documents. The trading partner definitions are crucial to the operation of the transaction document framework, as supplied. You may also need to create configurations (for example FTP inbound or outbound configurations) according to the transport protocols you have agreed with each trading partner.

You can enter basic contact details for each trading partner, although this is mainly for documentary purposes. Much more important to the transaction document processing framework are the data interchange attributes, linked directories, maps and configurations and the outbound numbering.

# **Remember:**

- " For EDI documents, the DISCOVER\_EDI activity uses several of the data interchange attributes to identify the interchange trading partner definition that corresponds to values in the EDI document.
- " The inbound and outbound processes use the linked directories associated with the trading partner (using the built-in variables) to determine directories

used to receive and process inbound documents and to prepare and send outbound documents and for archiving completed inbound or outbound documents.

- " The linked configurations may be used (via the built-in variables) to execute the particular transport arrangements agreed with the trading partner.
- " LANSA Composer uses the transformation maps linked to a trading partner (along with data interchange attributes specified for the map) to identify the transformation map that applies to a transaction document in the FIND\_TPMAP activity.
- " When you register a pending outbound transaction document via the supplied APIs or using the supplied TXDOC\_REGOUTBND or TXDOC\_REGOUTX12 activities, LANSA Composer uses (and updates) the information on the Outbound Numbering tab to determine the control numbers used for the outbound document. The TXDOC\_ALLOCCTRL activity also uses this information (but it is not used in the supplied processes).

# 5. Copy and modify the supplied processing sequences as required

Although you may use the inbound and outbound processes as supplied, more commonly you will copy them and adapt to your own circumstances. For example, as supplied, the inbound process makes simple assumptions about the type of transport to be used that may not be appropriate for your trading environment.

# Adapting your own Solution to Use the Transaction Document Register

The transaction document processing framework supplied with LANSA Composer may not exactly suit every circumstance or organisation. You are, of course, free to use LANSA Composer's transport, transformation and orchestration features to design and implement your own transaction document processing solution.

If you do, you may still opt to use the transaction document register and the LANSA Composer Document Manager application to record and manage your inbound and outbound transaction document flows.

In order to adapt an existing solution to use the transaction document register, there are just two Activities that you need to add to your transaction document processing solution:

- 1. The TXDOC\_REGISTER activity registers an in-coming or out-going transaction document in the transaction document register. Normally this should be done as soon as the document becomes available to the process and before significant processing is done for it.
- 2. After registering the document, and as the processing sequence proceeds to process the transaction document, you can record its progress and update the status in the transaction document register using the TXDOC\_STATUS activity.

If you use these activities appropriately, you will be able to use the LANSA Composer Document Manager to enquire upon and manage your inbound and outbound transaction document flows (although some features may not apply). For more information, refer to:

4.2 LANSA Composer Document Manager

# 4.1.3 Document Types and Document Standards

LANSA Composer's *Document types* and *Document standards* definitions describe certain characteristics of transaction documents so that inbound and outbound document processing can recognize them, validate them and invoke the appropriate processing sequence to process them.

A core set of *Document types* and *Document standards* definitions are supplied with LANSA Composer for recent EDIFACT and X12 document standards, but you can extend these with your own definitions according to your organization's requirements and trading relationships.

# **Document Types**

Document type definitions identify categories of transaction documents of the same type or format. This is necessarily an imprecise definition because the nature of the categories can be subject to organisational or application requirements. However, by way of example, the document types supplied with LANSA Composer include:

- EDI (all EDI types, whether they be EDIFACT or X12 or any of the many versions that apply to each)
- CSV (comma-separated-value formatted text files)
- XML (extensible markup language of any schema)

LANSA Composer's document type definitions serve these main purposes:

- 1. They provide the link between the transaction document file extension and the LANSA Composer document type definition that matches it (when using the DISCOVER\_DOC activity).
- 2. They provide a soft-coded link to the processing sequence that is used to process inbound or outbound transaction documents of the matching document type (when using the DISCOVER\_DOC or DISCOVER\_MAP activities).
- 3. They serve as a logical container or parent for document standard definitions (see below).
- 4. The document type for a transaction document file can be written to LANSA Composer's transaction document register when using the TXDOC\_REGISTER, TXDOC\_REGOUTBND, TXDOC\_REGOUTX12 and TXDOC\_STATUS activities. This supports categorization and interrogation of the transaction document register by document type using

the LANSA Composer Document Manager application.

For more information on creating and maintaining Document Type definition, refer to:

Document Type Maintenance

# **Document standards**

Document standard definitions belong to a document type and more precisely identify the type of information in transaction documents associated with the document standard. For example, LANSA Composer supplies document standards belonging to the document type EDI that identify the standard (X12 or EDIFACT) and the particular version of the standard applying to the EDI document.

LANSA Composer's document standard definitions serve these main purposes:

- 1. For an EDI (X12 or EDIFACT) transaction document, they provide the link between the document standard and version codes contained in the transaction document itself and the LANSA Composer document standard definition that matches it (when using the DISCOVER\_EDI activity).
- 2. For an EDI (X12 or EDIFACT) transaction document, they identify the SEF (standard exchange format) file that is used to validate EDI transaction document files that match the document standard (when using the DISCOVER\_EDI activity).

For more information on creating and maintaining Document Standard definitions, refer to:

Document Standard Maintenance
## **Document Type Maintenance**

To reach the Document Types, expand *Administration and Housekeeping* in the *Navigator* and then select *Document Types*.

You can use the Instance Lists to view and select from the available Document Types. Using the menu commands, toolbar buttons or right-click menus you can:

- Create, copy and delete Document Types
- Print Document Type definitions
- Display and/or revise the details, attachments or notes for the Document Types

## **Document Type Details**

ID	EDI
Description	Electronic data interchange
Document qualifier	
Document sequence	
Document file extension	edi
Document secondary extension	
Document name mask	
Inbound processing sequence	TXDOC_IEDI010
Outbound processing sequence	TXDOC_OEDI010

The following are the details that can be entered for a Document Type definition:

**ID** A unique identifier for the Document Type.

**Description** Text description for the Document Type.

**Group** This is a general purpose value that can be used to group various document types according to the context in which they are used. The DISCOVER\_DOC activity (if used) will match this value against a user-entered value for parameter DOCTYPEGROUP.

When processing a document, only those document type with the same Group as the one specified will be considered. When not

entered, a value of blank is assumed.

**Sequence** Sequence of processing. The DISCOVER\_DOC activity (if used) will compare incoming documents against Document types with smaller Document sequence number first. Once, a match is found, comparison stops and the remaining Document types are not considered.

The behaviour of the DISCOVER\_DOC activity is undefined if there are two document types with the same Group and Sequence number.

**Document**<br/>fileThe file extension that applies to this document type. The<br/>DISCOVER\_DOC activity (if used) will use this value to<br/>determine the document type for a document based on the file<br/>extension of the in-coming document.

For example, the file 'myfile.order.xml' has a file extension of 'xml' and could be eligible to be matched to a document type definition with that extension.

**Document**<br/>secondary<br/>extensionThis field may be used for extended file extension matching for<br/>documents that use more than one file extension in their name -<br/>for example, 'myfile.order.xml' If not blank, the<br/>DISCOVER\_DOC activity (if used) will use this value to<br/>determine the document type for a document based on the second

For example, the file 'myfile.order.xml' has a secondary file extension of 'order' and could be eligible to be matched to a document type definition with that secondary extension.

extension (the second from the right) of the incoming document.

**Document name mask** A string to be compared against the filename. If not blank, the DISCOVER\_DOC activity (if used) will use the value to determine the document type for a document based on the filename of the in-coming document. If there are more than two extensions, the third extension onwards will be included in the filename comparison.

In constructing a Document name mask, the following placeholder characters may be used:

? : will match any single character at the specified position of the filename

\*: will generically match any/all remaining characters in the filename

**Examples:** 

\* - Will match any file (equivalent to omitting this parameter value)

- will match files whose names begin with 'ab' ab\*

???d\* - will match all files whose names contain 'd' in the 4th position

Comparison is case insensitive.

Inbound Specifies the processing sequence that processes inbound processing transaction documents of this document type.

#### sequence

The DISCOVER DOC activity (if used) will return this identifier in the DOCPSEQ output parameter, such that the containing processing sequence may invoke the appropriate processing for the discovered document type of the incoming document.

Similarly the DISCOVER MAP activity can be used to discover this processing sequence identifier (providing the transformation map has been linked to a document type and standard in the Data Interchange attributes for the map).

Outbound Specifies the processing sequence that processes outbound processing transaction documents of this document type.

#### sequence

The DISCOVER MAP activity can be used to discover this processing sequence identifier (providing the transformation map has been linked to a document type and standard in the Data Interchange attributes for the map).

## **Document Standard Maintenance**

To reach the Document Standards, expand *Administration and Housekeeping* in the *Navigator* and then select *Document Standards*.

You can use the Instance Lists to view and select from the available Document Standards. Using the menu commands, toolbar buttons or right-click menus you can:

- Create, copy and delete Document Standards
- Print Document Standards definitions
- Display and/or revise the details, attachments or notes for the Document Standards

## **Document Standard Details**

Document type	EDI (Electronic data interchange)
Standard	X12
Version	004020
Description	X12.4020
Match agency	x
Match version	004020
SEF file name	X12_4020_STD.xml

The following are the details that can be entered for a Document Standard definition:

Document	Specifies the Document Type to which this document standard		
Туре	belongs. When creating a new document standard, you can		
	choose from a drop-down list of available Document Types. You must create a Document Type before you can create a Document Standard for that document type.		
Standard	Specifies a name by which this document standard will be known within LANSA Composer transaction document processing and in the LANSA Composer Document Manager.		
Version	Specifies a name by which this document standard version will be known within LANSA Composer transaction document processing and in the LANSA Composer Document Manager.		

**Description** Text description for the Document Standard.

- Match<br/>agencySpecifies the agency value in the EDI transaction document that<br/>matches this document standard and version. This value (along<br/>with the *Match version*) is used by the DISCOVER\_EDI activity<br/>to determine the Document Standard definition in LANSA<br/>Composer that applies to the subject EDI transaction document<br/>file.
- Match<br/>versionSpecifies the version value in the EDI transaction document that<br/>matches this document standard and version. This value (along<br/>with the Match agency) is used by the DISCOVER\_EDI activity<br/>to determine the Document Standard definition in LANSA<br/>Composer that applies to the subject EDI transaction document<br/>file.
- SEF file Specifies the name of the SEF (standard exchange format) file that is used by the DISCOVER\_EDI activity to validate EDI transaction document files. The DISCOVER\_EDI activity will expect to find the SEF file (if provided) in the directory for SEF files specified in *System Settings*.

# **4.1.4 Trading Partner support for Transaction Document Processing**

The transaction document processing framework, as supplied, makes significant use of Trading Partner definitions and features. Both the inbound and outbound processes process transaction documents by Trading Partner, in the first place.

Amongst the base attributes for a trading partner are values that let you identify specific processing sequences to perform receive and send processing for that trading partner. This optional feature provides additional flexibility for you to tailor your transport mechanisms to your particular trading relationships. If specified, these processes are invoked from the standard "model" inbound and outbound processes.

Furthermore, Trading Partner definitions provide several crucial links:

- To the directories used to hold the inbound or outbound transaction documents through the various stages of their processing (using the Linked Directories feature of a Trading Partner definition)
- To the configurations used for transport and other activities that form part of the transaction document process (using the Linked Configurations feature of a Trading Partner definition)
- To the Transformation Maps used to receive inbound transaction document data or prepare outbound transaction document data for a given document type, standard, transaction type and direction (using the Linked Maps feature of a Trading Partner definition)

In addition, the Trading Partner Data Interchange Attributes provide certain values that facilitate the recognition of the source and / or the target of certain transaction document types and specify aspects of how they are to be processed.

## **Trading Partner Data Interchange Attributes**

LANSA Composer defines and stores a set of data interchange attributes for each Trading Partner. These are defined to accommodate some of the attributes commonly used in EDI and other forms of data interchange.

While LANSA Composer pre-defines fields to hold these attributes, the program does not enforce the entry of the fields, nor does it validate values entered. To permit maximum implementation flexibility, LANSA Composer leaves the use and values of many of these attributes to you according to your own circumstances and conventions.

Some of the values are, however, important for the operation of the transaction document framework as supplied. These include:

Archive file prefix(inbound and outbound)	These values determine the naming used when the inbound or outbound process archives a transaction document file that has been successfully processed.	
Inbound receiver id Inbound sender id	For an inbound EDI X12 transaction document file these values are matched against corresponding fields in the ISA segment of the EDI document. A successful match enables LANSA Composer to identify the Trading Partner definition to which the EDI transaction applies.	
	This matching to a Trading Partner is performed in the DISCOVER_EDI activity and a successful match is necessary for further processing.	
Outbound receiver id Outbound sender id	For an outbound EDI X12 transaction document file these values are matched against corresponding fields in the ISA segment of the EDI document. A sucessful match enables LANSA Composer to identify the Trading Partner definition to which the EDI transaction applies.	
	This matching to a Trading Partner is performed in the DISCOVER_EDI activity and a successful match is necessary for further processing.	
EDI validation	The DISCOVER_EDI activity, having matched the EDI transaction document to a Trading Partner, uses the value of this attribute for the Trading Partner to determine whether	

	and how to perform EDI document validation.
	If the value is set to <i>None</i> , no EDI document validation is performed.
	If the value is set to <i>Errors</i> , EDI document validation is performed and any validation errors result in document processing ending in error. Normally, if you are able to correct the EDI transaction document, processing may be restarted and the transaction document process run to normal completion.
	If the value is set to <i>Warnings</i> , EDI document validation is performed but document processing will continue irrespective of whether validation errors are found.
EDI split maximum transactions	These values are passed to the EDI_SPLIT activity for an inbound EDI transaction document file to determine whether and how the transaction document file is split into multiple files containing a subset of the transactions in the original
EDI split ignore carriage returns	document.

The trading partner data interchange attributes are used in the transaction document processing framework, or can be used in your own custom transaction document processing solution.

The attribute values are accessible within a processing sequence by using builtin variables such as \*tradingpartner.edi.iap (for the inbound archive file prefix). Refer to Trading Partner (\*TRADINGPARTNER) Built-in Variable Qualifiers for more information.

## 4.1.5 Activities for Transaction Document Processing

The supplied transaction document processing framework is typical of many business process integration (BPI) solutions and makes use of a wide variety of the Activities that are supplied with LANSA Composer.

However, there is a subset of the supplied Activities supplied that are very specifically intended for and associated with the transaction document processing framework. These are Activities that:

- Discover document and content type, including Activities specific to EDI document discovery.
- Establish connections between transaction document files and the Trading Partner and Transformation Maps that should be used to process them.
- Register transaction document files in LANSA Composer's transaction document register and update their status as document processing proceeds.
- Invoke customer-defined processing to import or export transaction document data to or from application database tables.

This group of supplied Activities are listed in:

Transaction Document Processing

As supplied, the Processing Sequences that implement the transaction document processing framework use most of these Activities.

However, if you either modify the supplied solution or create your own, you are not compelled to use the supplied activities if it is inappropriate, in your particular circumstances to do so.

For example, the supplied TXDOC\_INBOUND Processing Sequence uses the DISCOVER\_DOC activity to associate a transaction document file extension with the subsidiary Processing Sequence that processes that type of transaction document file. This provides the maximum flexibility in the supplied solution, but in many cases, with knowledge of your specific organization's requirements, it may be equally valid to make assumptions about the type of transaction document files that you receive from a given Trading Partner.

There are only two Activities that are usually essential to any solution in which you wish to make use of LANSA Composer's transaction document register. They are the TXDOC\_REGISTER and TXDOC\_STATUS Activities. For more information, refer to:

Adapting your own Solution to Use the Transaction Document Register

## **4.1.6 Processing Sequences for Transaction Document Processing**

LANSA Composer's transaction document processing framework includes sets of processing sequences that have been pre-built for processing both inbound and outbound transaction document flows.

You can use these unchanged or you can copy and adapt them to the specific requirements of your organization. The following topics provide further information:

Modify the Supplied TXDOC\_DATABASE Database Configuration Copy the Supplied Processing Sequences Before Use The Inbound Process The Outbound Process

# Modify the Supplied TXDOC\_DATABASE Database Configuration

As supplied, the inbound and outbound processes use the TXDOC\_DATABASE configuration when executing a Transformation Map involved in the transaction document process.

It is intended that you should modify this TXDOC\_DATABASE configuration as required to address your own database. Alternatively, you can modify your own copies of the inbound and outbound processes to use your own database configuration.

🐻 Database configuration : Deta	ils (TXDOC_DATABASE-Transaction document database configuration)	
Details 🚺 Notes 🕢 Audit		
ID Description	TXDOC_DATABASE Transaction document database configuration	<u>arr</u> <u>S</u> ave
Status	Active	Refresh
Database connection string	jdbc:as400://LANSA01/LICLICLIB;prompt=false;naming=sql;data truncation=fals	
Or		
Database ID		
Database driver class	com.ibm.as400.access.AS400JDBCDriver	
Or		
Database driver ID		
Database user	LICPGMLIB	
Database password	*****************	
Commit automatically	Yes	
Transaction isolation	None	

## **Copy the Supplied Processing Sequences Before Use**

The sets of Processing Sequences supplied with LANSA Composer for processing inbound and outbound transaction document flows:

- Have names beginning with TXDOC\_;
- Are supplied with their *Supplied processing sequence* flag set to *Yes*.

The latter point means you may not change these definitions, although you may open them in *read-only* mode and/or print their definitions. There is no supported means to turn off this flag such that you can modify the Processing Sequences.

This is deliberately so. The supplied Processing Sequences may be updated and replaced with later versions of LANSA Composer. If LANSA Composer allowed you to change them, then your changes would be lost when you upgrade to a new version.

If you intend to use these Processing Sequences, and particularly if you wish to modify them, you are strongly advised to make your own copies and use or modify the copies. When doing so, please refer to the following notes.

## **Copying the Supplied Processing Sequences**

The supplied inbound and outbound processes are designed to be modular - that is, they are broken down into more than one Processing Sequence with the main Processing Sequence (TXDOC\_INBOUND or TXDOC\_OUTBOUND) calling the subsidiary Processing Sequences as required.

If you copy the Processing Sequences, you will need to modify any references to subsidiary Processing Sequences to make sure that your copies are called. This may involve one or other of the following:

1. Modifying a Document Type definition

The Document Type definition includes a reference to the Processing Sequence that processes transaction documents of the type. For example, as supplied, the 'EDI' document type is configured to use Processing Sequence TXDOC\_IEDI010 to processing inbound transaction documents, or TXDOC\_OEDI010 to processing outbound transaction documents.

The supplied TXDOC\_INBOUND and TXDOC\_OUTBOUND Processing Sequences uses these values to determine which Processing Sequence to execute for a transaction document file.

If you have copied supplied Processing Sequences that are specified in a

Document Type definition (EDI or another type), then you should modify the Document Type definition to call your copied version.

For more information on modifying the Document Type definition, refer to Document Type Maintenance.

2. Modifying references within the Processing Sequence definition

In other cases, references to called Processing Sequences are embedded directly in the calling Processing Sequence.

For example, for an EDI document, the supplied TXDOC\_IEDI010 Processing Sequence calls TXDOC\_IEDI020 which in turn may call TXDOC\_IEDI090.

In these instances, you need to open your copy of the *calling* Processing Sequence in the *Processing Sequence Editor* and directly change the reference to the *called* Processing Sequence.

For more information on editing a Processing Sequence, refer to Use the Processing Sequence Editor.

## **The Inbound Process**

Note 1: the following descriptions relate to the inbound processes as supplied with LANSA Composer. Naturally it does not take account of modifications that may have been made to suit a particular installation.

Note 2: you are strongly advised to make your own copies of the supplied Processing Sequences and use or modify the copies. For further information, please refer to Copy the Supplied Processing Sequences Before Use.

The supplied inbound process is designed to be modular - that is, it is broken down into more than one Processing Sequence with the main Processing Sequence calling the subsidiary Processing Sequences as required.

The main inbound Processing Sequence is:

TXDOC\_INBOUND Processes inbound transaction documents

This Processing Sequence can be run for one named Trading Partner, or for all eligible Trading Partners according to the value of the corresponding parameter. It performs the following main steps:

• Receives transaction document files from each Trading Partner

Receiving transaction document files is accomplished by calling a separate processing sequence, **TXDOC\_RCV**. This processing sequence, as supplied, includes provision for executing processing sequences that are specific to one or more trading partners, providing the possibility of tailoring the inbound transport for each trading partner.

• For each transaction document file, TXDOC\_INBOUND uses the DISCOVER\_DOC Activity to determine the document type and then to call the inbound Processing Sequence associated with that document type.

LANSA Composer is supplied with inbound documents of types EDI and XML configured to be processed using the following Processing Sequences:

**TXDOC\_IEDI010** Processes inbound EDI transaction document files

This Processing Sequence calls the following subsidiary

#### Processing Sequences: TXDOC\_IEDI020 TXDOC\_IEDI090

TXDOC\_IXML010 Processes inbound XML transaction document files

## **The Outbound Process**

**Note 1:** the following descriptions relate to the outbound processes as supplied with LANSA Composer. Naturally it does not take account of modifications that may have been made to suit a particular installation.

**Note 2:** you are strongly advised to make your own copies of the supplied Processing Sequences and use or modify the copies. For further information, please refer to Copy the Supplied Processing Sequences Before Use.

The supplied outbound process is designed to be modular - that is, it is broken down into more than one Processing Sequence with the main Processing Sequence calling the subsidiary Processing Sequences as required.

The main outbound Processing Sequence is:

TXDOC\_OUTBOUND Processes outbound transaction documents

This Processing Sequence can be run for one named Trading Partner, or for all eligible Trading Partners according to the value of the corresponding parameter. Other parameters allow you to select the document type, standard and/or transaction types that it will process. It performs the following main steps:

- Identifies the outbound Transformation Maps linked to each Trading Partner (that also match the other parameter values if specified)
- For each, it calls the outbound Processing Sequence associated with the document type for the Transformation Map.

LANSA Composer is supplied with outbound documents of types EDI and XML configured to be processed using the following Processing Sequences:

**TXDOC\_OEDI010** Processes outbound EDI transaction document files

TXDOC\_OXML010 Processes outbound XML transaction document files

In each of those processing sequences, once the outbound document file(s) have been prepared, the supplied **TXDOC\_SEND** processing sequence is executed to send the document file to the trading partner. This processing sequence, as supplied, includes provision for executing processing sequences that are specific to one or more trading partners, providing the possibility of tailoring the outbound transport for each trading partner.

Note that, as supplied, the **TXDOC\_SEND** processing sequence is also used to serve the *Resend* capability in the LANSA Composer Document Manager application.

## **4.1.7 Transformation Maps for Transaction Document Processing**

Transformation maps are almost always an integral part of a transaction document processing solution:

- For inbound processing, they are typically used to map the data contained in the inbound transaction document into the application database, either directly or via intermediary "staging" database tables.
- For outbound processing, they are typically used to extract data from the application database and map it into the outbound transaction document.

The following topics contain more information about the use of Transformation Maps in the transaction document processing framework:

Model Transformation Maps for Transaction Document Processing

Transformation Map Data Interchange Attributes

For information about pre-built transformation maps supplied for use with the transaction document processing framework, refer to:

4.1.8 Pre-built EDI X12 solution components

For general information concerning Transformation Maps, refer to:

**Transformation Maps** 

The Mapping Tool

**Note** If you intend to use Altova MapForce to create mappings involving EDI X12, EDIFACT, HL7 or certain other specialized transaction standards, you may need to install additional configuration files for that transaction standard. These are not installed by the LANSA Composer client installation.

The additional installers for EDI X12 and EDIFACT for the shipped version of Altova MapForce are provided on the LANSA Composer client media.

Configuration files for other versions or standards may be downloaded from the Altova web-site at

http://www.altova.com/components\_mapforce.html.

## Model Transformation Maps for Transaction Document Processing

The pre-built processing sequences that are central to the transaction document processing framework assume a particular pattern for the transformation maps that it invokes and LANSA Composer supplies the following "model" transformation maps that match this pattern:

- TXDOC\_MODEL\_IN (Transaction document inbound model map)
- TXDOC\_MODEL\_OUT (Transaction document outbound model map)

These maps cannot be used directly but are supplied to serve as a model from which you can create your own maps that reference the database and the transaction document types involved in your organisation's transaction document processing.

The key thing about these maps is that they define the number, order and meaning of the parameters that your maps will be assumed to receive when invoked by the transaction document processing framework, as supplied. Any maps you create will need to accept the same parameters in order to be compatible with the supplied transaction document processing sequences. For inbound maps, the expected parameters are:

#### Name (\*)

#### Description

TXDOC\_MODEL\_IN\_1(TextfileSourceFilename) Specifies the path to the inbound transaction document file.

(Adding a file system source component (for example: XML, text, EDI, XBRL or Excel) to your map results in such an input parameter.)

Specifies the transaction document envelope number that identifies the

#### TXDOC\_MODEL\_IN\_2 (DocNumberSourceParameter)

transaction document in the transaction document register. This value is returned by the TXDOC\_REGISTER activity.

(Your map may or may not need to reference this value, but you must define it as an input parameter in your map in any event in order for your map to be compatible with the processing sequences, as supplied, used by the transaction document processing framework.)

Specifies a LANSA Composer database configuration name that identifies the database configuration used to connect to the target database for the map.

(Adding a database target component to your map results in such a parameter.)

(\*) Note that your map does not have to (and will not) use the same parameter names as shown - it is the number, order and meaning of the transformation map parameters that is important for this purpose.

For outbound maps, the expected parameters are:

#### TXDOC\_MODEL\_IN\_3 (TXDOC\_DATABASETargetConnection)

#### Name (\*)

#### TXDOC\_MODEL\_OUT\_1 (TXDOC\_DATABASESourceConnection)

#### Description

Specifies a LANSA Composer database configuration name that identifies the database configuration used to connect to the source database for the map.

(Adding a database source component to your map results in such a parameter.)

#### Specifies the transaction document envelope number that identifies the transaction document in the transaction document register. This value is returned by the TXDOC\_REGOUTBND or TXDOC\_REGOUTX12 activities or equivalent APIs and by the FOR\_EACH\_TXDOCO iterator activity.

(Your map may or may not need to reference this value, but you must define it as an input parameter in your map in any event in order for your map to be compatible with the processing sequences, as supplied, used by the transaction document processing framework.)

Specifies the path to the outbound transaction document file to be created.

#### TXDOC\_MODEL\_OUT\_2 (DocNumberSourceParameter)

#### TXDOC\_MODEL\_OUT\_3 (TextfileTargetFilename)

(Adding a file system target component (for example: XML, text, EDI, XBRL or Excel) to your map results in such an input parameter.)

(\*) Note that your map does not have to (and will not) use the same parameter names as shown - it is the number, order and meaning of the transformation map parameters that is important for this purpose.

## **Transformation Map Data Interchange Attributes**

LANSA Composer defines and stores a set of data interchange attributes for each Transformation Map. These are defined to accommodate attributes that are used by the transaction document processing framework as follows:

Map direction	Specifies the direction (inbound or outbound) of the transaction document processing flow to which this map applies.		
	When the map is linked to a Trading Partner, this attribute is referenced to identify qualifying transformation maps when using the FIND_TPMAP activity with the DIRECTION parameter.		
Map standard	Specifies, by selection from a drop-down list, the document type and the document standard to which this map applies.		
	When the map is linked to a Trading Partner, this attribute is referenced to identify qualifying transformation maps when using the FIND_TPMAP activity with the STANDARD parameter.		
Map transaction type	Specifies the transaction type to which this map applies. For example, '850' for an EDI X12 purchase order transaction or 'ORDERS' for an EDIFACT order transaction.		
	When the map is linked to a Trading Partner, this attribute is referenced to identify qualifying transformation maps when using the FIND_TPMAP activity with the TRANSACTIONID parameter.		
Import processor	Specifies the name of the import processor that imports inbound transaction data into an application database after it has been received, typically into staging database tables, by this transformation map.		
	This name should be passed to the TXDOC_IMPORT activity in order to execute the import step for an inbound transaction document after executing the associated transformation map.		
Export processor	Specifies the name of the export processor that extracts transaction information from an application database and writes		

it, typically into staging database tables, to be read by this transformation map to prepare an outbound transaction document.

This name should be passed to the TXDOC\_EXPORT activity in order to execute the export step for an outbound transaction document before executing the associated transformation map.

The transformation map data interchange attributes are used in the transaction document processing framework, or can be used in your own custom transaction document processing solution in the following ways:

• Where applicable, the attribute values are accessible within a processing sequence by using built-in variables such as *\*transform.edi.imp* and *\*transform.edi.exp* for the import and export processor names. You must first have assigned a transformation map identifier to the *\**transform built-in variable before you can access its attribute values in this way.

Refer to Transformation Map (\*TRANSFORM) Built-in Variable Qualifiers for more information.

- LANSA Composer refers to some of the attribute values to identify qualifying transformation maps when using the FIND\_TPMAP activity.
- Some of these and other attributes relating to the document type and standard can be retrieved in a processing sequence using the DISCOVER\_MAP activity.

## **Import and Export Processors**

Note that the use of import and export processors is not mandatory. These attributes and this capability are provided to support the use of intermediary "staging" database tables when processing inbound or outbound transaction documents. If you choose instead that your transformation maps directly read from and write to your application database tables, then you may not need to separately define or execute an import or export step.

For more information about defining import and export processors, refer to:

4.1.9 Application program interfaces (APIs)

## 4.1.8 Pre-built EDI X12 solution components

LANSA Composer is supplied with a limited number of pre-built transformation maps, staging database tables and export and import processors and document viewers for selected EDI X12 transactions.

These pre-built components may serve as a pro-forma solution for those transactions, but more especially as examples of a completed solution upon which you might base a solution for other types of transaction documents used by your organization.

The EDI X12 transactions and the associated pre-built components are listed below:

X12 Transaction Type	Inbound and/or Outbound Maps	Import and/or Export Processors and Document viewers	Staging Database Tables Supplied
810	O810_4010	n/a	Yes (EDI810*)
850	I850_4010	EDI850IMP EDI850VWR	Yes (EDI850*)
855	O855_4010	n/a	Yes (EDI855*)
997	I997_4010 O997_4010	EDI997EXP EDI997IMP EDI997VWR	Yes (EDI997*)

For more information on the types of components mentioned above and the role they play in the transaction document processing framework, please refer to:

4.1.7 Transformation Maps for Transaction Document Processing Import and Export Processors

EDI Document Viewers

Source code for the pre-built export and import processors and document viewers mentioned above is included in the LANSA Composer development package. For information on finding and installing the LANSA Composer development package, please refer to Before you begin your Custom Activity.

**Note** the supplied import and export processors read from or write to sample application database tables installed with LANSA Composer. If you wish to adapt these supplied components for use with your own application, you will, at the least, have to replace the import and export processors with ones that reference your own application database.

## 4.1.9 Application program interfaces (APIs)

The supplied transaction document processing framework can be extended in several ways to provide extended support for specific transaction types and to better integrate into your own applications.

One of the ways in which LANSA Composer supports such extensibility is by providing application program interfaces (APIs) in the forms of ancestor components for your own LANSA components and by providing callable LANSA interfaces.

Refer to the following for more information:

Import and Export Processors

**EDI Document Viewers** 

**Registering a Pending Outbound Document** 

Register a Transaction Document and Update Its Status

## **Import and Export Processors**

You can write your own import and export processors to handle the identification selection, validation and cleansing of data between your own application database and the intermediate staging database tables.

Refer to the following for more information:

Develop an import or export processor

Connect your import or export processor to the transaction document processing framework

NB: The need for import and export processors is predicated on your transaction document processing solution using intermediate "staging" database tables as the target for inbound transformation maps and/or the source for outbound transformation maps. This solution insulates your application from the interfaces that the transaction document processing solution opens to the outside world. However, the use of "staging database tables may not be necessary in all cases. If your transformation maps write and/or read directly to or from your application database, then you may have no need to implement import and export processors.

#### Develop an import or export processor

Your import and export processors must be implemented as LANSA RDMLX components (re-useable parts) that will reside and execute in the same LANSA system and partition as LANSA Composer.

For general considerations about integrating your custom LANSA code with your LANSA Composer solution refer to:

Appendix E. Using LANSA Composer with LANSA Applications

You must use DXXBASIMP as the ancestor of your import processor component and DXXBASEXP as the ancestor of your export processor component.

Before you begin development, you must prepare your development environment. You will need a correctly licensed and configured Visual LANSA development environment at the level defined in Install LANSA Composer.

• Development of your custom import and export processors must be performed in a partition that is enabled for full RDMLX.

• You must import the LANSA Composer development package into the partition in which you will develop your import and export processors. This provides the DXXBASIMP and DXXBASEXP ancestor classes that you will need to compile your processors

The LANSA Composer development package can be found in the folder DEV\Import on your LANSA Composer software DVD.

## Connect your import or export processor to the transaction document processing framework

When you have built your import or export processor and deployed it to the LANSA Composer server environment, you must connect it to the transaction document processing framework so that LANSA Composer knows for which type of transactions it should invoke your processor(s).

This is done by specifying the name of the import or export processor in the appropriate data interchange attribute field of the Transformation Map(s) that process the transaction type to which it applies.

For more information, refer to Transformation Map Data Interchange Attributes.

## **EDI Document Viewers**

You can create specialized document viewers that plug-in to the LANSA Composer Document Manager for viewing transaction-specific data.

Contact your LANSA Composer supplier for more information about this type of extension.

## **Registering a Pending Outbound Document**

Typically the need to create and send an outbound transaction document to one of your trading partners is triggered by an event in your application such as the entry of a purchase order or the fulfillment of a sales order.

When such an event occurs, your application needs to notify the LANSA Composer transaction document processing framework such that LANSA Composer can subsequently generate and send the outbound document.

Your application can do this by registering a pending outbound transaction document. Subsequently the outbound process detects and processes the pending outbound document.

There are three means provided for you to register a pending outbound transaction document, described under the following headings:

Registering a pending outbound document using the TXDOC\_REGOUTBND activity

Registering a pending outbound document using the LANSA function API

Registering a pending outbound document using the LANSA component API

For a more general API that is capable of registering both inbound AND outbound documents, you may wish to refer also to:

Register a Transaction Document and Update Its Status

## **Registering a pending outbound document using the TXDOC\_REGOUTBND activity**

The simplest case is where you have an existing LANSA Composer Processing Sequence which is responsible for the event that triggers the need to generate and send an outbound transaction document. In that event, your process can include the supplied TXDOC\_REGOUTBND activity to register the pending outbound transaction document and no programming is required.

For more information on this activity, refer to TXDOC\_REGOUTBND.

LANSA Composer also supplies a specialized form of this activity, TXDOC\_REGOUTX12, for registering pending outbound EDI X12 transactions documents.

## **Registering a pending outbound document using the LANSA** function API

**Note** This section assumes sound knowledge of the applicable LANSA development techniques. Refer to LANSA product documentation for detailed information on application development with LANSA.

LANSA Composer provides the DXXREGO function for registering a pending outbound transaction document from a LANSA application.

- Your LANSA application can call DXXREGO using the CALL RDML command with PROCESS(\*DIRECT). Function DXXREGO must execute on the LANSA Composer server. Your application may, however, execute it via the LANSA SuperServer CALL\_SERVER\_FUNCTION built-in function.
- Your application can pass the necessary variable information using the LANSA exchange list as follows:

Name	Description	Туре
DXTPII	Trading partner internal or external id	A(32)
Either DXXDCTID DXXDSTID DXXDSTVR Or	Document type name Document standard name Document standard version	A(15) A(15) A(15)
DXXDSTII	Transaction id (eg: '850', 'INVOIC')	A(32) A(6)
DXXTEST	Production (P) or test (T) indicator	A(1)
DXXKEY01	Staging file key 1 (eg: Company code)	A(32)
DXXKEY02	Staging file key 2 (eg: Division code)	A(32)
DXXKEY03	Staging file key 3 (eg: Department code)	A(32)
DXXKEY04	Staging file key 4 (eg: Invoice number)	A(32)
DXXKEY05	Staging file key 5 (eg: other)	A(32)
DXXKEY06	Staging file key 6 (eg: other)	A(32)

• Your application can receive the result of the request via the exchange list as follows:

Name	Description	Туре
DXRESULT	Result code (OK, ER)	A(2)
DXXENVID	Transaction document envelope id	S(9, 0)

**Note** If you have imported the LANSA Composer development package into your Visual LANSA development environment, the definitions of the fields named above will already be present in your LANSA repository.

## Registering a pending outbound document using the LANSA component API

**Note** This section assumes sound knowledge of the applicable LANSA development techniques. Refer to LANSA product documentation for detailed information on application development with LANSA.

LANSA Composer provides the DXXREGOUT re-useable part that provides several interfaces for registering a pending outbound transaction document from a component-based LANSA application.

The application can choose from two interfaces:

- 1. Method *dxSimpleDocument* allows the application to create and register a single document envelope containing a single interchange, group, and transaction message in one call
- 2. An alternate set of methods allows the application to create and register a more complex transaction document containing multiple interchanges, groups and/or transaction messages:

dxNewDocument1 OR dxNewDocument2 dxNewInterchange (optional) dxNewGroup (optional) *dxNewMessage* 

*dxRegisterDocument* 

When using either interface style, the application can optionally handle the following events:

*dxValidationErro*r

dxPreInsert2IN

dxPreInsert3GP

dxPreInsert4MS

The latter three events offer an opportunity to customize the transaction data generated by the API before it is written to the transaction document database.

**Note** Once you have imported the LANSA Composer development package into your Visual LANSA development environment, you can use the feature help within the Visual LANSA IDE to review the detail of the methods and events exposed by the DXXREGOUT component.

## **Register a Transaction Document and Update Its Status**

In the previous section (Registering a Pending Outbound Document) a set of APIS were described for registering a pending outbound transaction document that can then be picked up and processed by the usual outbound processes. This is by far the most common case where you may need to update LANSA Composer's Transaction Document register from your own application code.

However, for more advanced uses, LANSA Composer provides a more general API that is capable of both:

- Registering a transaction document (whether inbound OR outbound); and
- Updating the status of a registered transaction document

These two functions correspond directly to the functions of the TXDOC\_REGISTER and TXDOC\_STATUS activities. (Note that they do not provide the EDI-particular features that are provided by the APIs described in Registering a Pending Outbound Document.)

**Note** This section assumes sound knowledge of the applicable LANSA development techniques. Refer to LANSA product documentation for detailed information on application development with LANSA.

LANSA Composer provides the DXXREGDOC re-useable part that provides the interfaces through two methods:

- 1. Method *dxDocRegister* allows the application to register a transaction document and is functionally equivalent to the supplied TXDOC\_REGISTER activity.
- 2. Method *dxDocStatus* allows the application to update the status of a specified transaction document and is functionally equivalent to the supplied TXDOC\_STATUS activity.

**Note** Once you have imported the LANSA Composer development package into your Visual LANSA development environment, you can use the feature help within the Visual LANSA IDE to review the detail of the methods and properties exposed by the DXXREGDOC component.
## 4.2 LANSA Composer Document Manager

For transaction document processing solutions that make use of LANSA Composer's built-in transaction document register (via the TXDOC\_REGISTER, TXDOC\_REGOUTBND, TXDOC\_REGOUTX12 and TXDOC\_STATUS activities), LANSA Composer provides a separate LANSA Composer Document Manager application.

The LANSA Composer Document Manager provides the ability to enquire upon and monitor the flow of inbound and outbound transaction documents.

The following topics provide more information about working with the LANSA Composer Document Manager:

4.2.1 Start the LANSA Composer Document Manager

4.2.2 Work with Transaction Documents

4.2.3 Work with Transaction Document Statistics

The LANSA Composer Document Manager also provides access to LANSA Composer's *Operations Console* and *Java Service Manager console*. For information on these refer to the following:

Operations Console Java Service Manager Console

## 4.2.1 Start the LANSA Composer Document Manager

You can start the LANSA Composer Document Manager in two ways:

• Using the shortcut provided on the Start menu

When you start the LANSA Composer Document Manager this way, you will see the *Connect to LANSA Composer Server* window to let you specify all the information necessary to make a connection to your LANSA Composer server. Refer to Connect to LANSA Composer Server for more information.

• From within the LANSA Composer client application by using the *Open LANSA Composer Document Manager* item from the *Tools* menu. You do not have to logon to your LANSA Composer server because LANSA Composer uses the details you used to start the LANSA Composer client application.

Once you have successfully connected, the LANSA Composer Document Manager *Start Here* window is displayed.



#### 4.2.2 Work with Transaction Documents

To work with Transaction Documents, expand *Files* in the *Navigator* and then select *Documents*. The Documents filter and Documents list will be displayed. You can then:

Filter Documents Work with the Documents List View Document Details View Document Processing History View other document details

#### **Filter Documents**

You can filter transaction documents by a wide variety of criteria that enable you to perform very specific searches.

🔽 Clear List	🔄 Reset 🛛 🕅 Search
Exchange trading partner	
Interchange trading partner	
Exchanges from	📝 29/01/2010 4:00:00 AM 💌
Exchanges to	
Exchange direction	Both
Exchange result code	All
Standard	All
Transaction type	850
Acknowledged?	All
Split status	All
Validation status	All
Production status	All
Staging key 1	
Staging key 2	
Staging key 3	
Staging key 4	
Staging key 5	
Staging key 6	
User field 1 value	
User field 2 value	

Remember to check the *Clear List* checkbox if you wish the filtered items to replace the existing list. If you uncheck the *Clear List* checkbox, then the items found by your search will be added to the existing list if not already there.

Click the Search button to perform your search and add the matching items to the Documents list.

#### Work with the Documents List

The Documents List displays the registered transaction documents that matched your search criteria. You can select an item to view its details in the bottom portion of the screen.

Document Id		Trading Partner	Direction	Split	Exchange Date/Time	Exchange Status	Doc	Document Location
+	421	TXDOC_EXCHANGE	Inbound	Child	2010-02-12 10:07:44.406	PR - Found transfo	EDI	/LANSA_COMPOSER_XF3PGMLIB/XF3
+	425	TXDOC_EXCHANGE	Inbound	Child	2010-02-22 11:54:45.596	OK - Successfully c	EDI	/LANSA_Composer_xf3pgmlib/xf3/TP/.
Ξ	1192	TXDOC_EXCHANGE	Inbound	Child	2010-05-29 13:44:29.223	OK - Successfully c	EDI	/LANSA_Composer_licpgmlib/lic/TP/TC.
	🖃 🍯 Trading par							
	+ 1192.1	000000159						
4								

The Documents list is displayed as a tree. For EDI documents, you can expand the top-level entries to select and review details that apply at the following levels:

- Trading partner agreements (or *interchanges*)
- Groups
- Transactions (or *messages*)

**Note** you can directly enquire upon and view lists of documents at these levels by expanding *Messages* in the *Navigator* and selecting the corresponding option. These options normally apply only to EDI transaction documents. They offer similar filtering and viewing capabilities to those supplied by the *Documents* option.

#### **View Document Details**

When you select a transaction document in the Documents List, you can click the *View* tab to view the registered details for the transaction document.

🔟 View 🛛 🔂 History 🔥 Validation Errors						
Document Id	1192	Refresh				
Processing sequence run number	Ø000000002192	O. View				
Exchange trading partner						
Exchange direction	Inhound	Print				
Exchange date/time	2010-05-20 12:44:20 222	Restart				
Exchange date/dille		bandon				
Exchange status	Successfully completed processing document					
Exchange file location	/LANSA_Composer_licpgmlib/lic/TP/TC_IPARTNER1/IDOC_ARCHIVE/idoc220-0001-0001.					
Exchange original file	/LANSA_Composer_licpgmlib/lic/TP/TXDOC_EXCHANGE/IDOC_PROCESS/idoc220-0001-0001.e					
Split status	Child					
Parent document id	1191					
Response to document id						
Document type	EDI					
Document size (bytes)	618					
Transaction cost (AUD)	.01000					

You can click the *View* button to display the Processing Sequence Log for the Processing Sequence Run that processed the transaction document. Click *Print* 

to print the Processing Sequence Log.

**Note** The Processing Sequence Log is not the same as the transaction document history. The latter can be seen by selecting the *History* tab.

If the processing for a transaction document ended in error, and you are able to correct the cause of the error, you can restart the processing for that document by clicking the *Restart* button. If it is not possible or appropriate to resume processing for the transaction document, click *Abandon* to mark the document as complete.

If the processing for an *outbound* transaction document has completed successfully, a *Resend* button is available. In the event that you need to resend the completed document to the exchange trading partner (and assuming the document is still available in the last recorded location shown), you may click *Resend* to accomplish that.

When you click *Resend*, LANSA Composer will execute the processing sequence named in the *"Resend" processing sequence* system setting to fulfil the request. As supplied, this is the TXDOC\_SEND processing sequence, but it may have been customised in your installation. That processing sequence, in turn, may invoke a processing sequence specifically associated with the trading partner for the purpose.

#### View Document Processing History

When you select a transaction document in the Documents List, you can click the *History* tab to view the processing history for the transaction document.

🔟 View 📃 History 🛕 Validation Errors 📗			
Registered transaction document     Discovered document type     Matched trading partner to document content     Validated document content successfully	Processing sequence run number Exchange log sequence Exchange log date/time	00000000002192           0000000009           2010-05-29 13:44:41.453	Refresh
Found transformation map for document     Transformation successful for document     Imported transaction data successfully     Acknowledgement completed for transaction	Exchange result code Exchange status Document type	OK Successfully completed processing documer EDI	
9 Successfully completed processing document	Exchange content type Exchange file location	850 /LANSA_Composer_licpgmlib/lic/TP/TC	

The transaction processing history consists of a line for each TXDOC\_REGISTER or TXDOC\_STATUS activity that was executed in the course of processing the transaction document.

#### View other document details

The Documents list is displayed as a tree. For EDI documents, you can expand the top-level entries to select and review details that apply at the following levels:

- Trading partner agreements (or *interchanges*)
- Groups
- Transactions (or *messages*)

**Note** you can directly enquire upon and view lists of documents at these levels by expanding *Messages* in the *Navigator* and selecting the corresponding option. These options normally apply only to EDI transaction documents. They offer similar filtering and viewing capabilities to those supplied by the *Documents* option.

If supported for the document type, standard and transaction type, you may also be able to view:

• Validation Errors

If available select the *Validation Errors* tab to display details of any validation errors detected for the Transaction Document. As supplied, this feature is available only for EDI documents that were validated in the DISCOVER\_EDI Activity and that have a suitable SEF file against which to validate.

• Document content

If available select the *Contents* tab to display formatted details of the transaction document content. As supplied, this feature is available only for selected EDI X12 transactions for which a viewer is supplied.

## 4.2.3 Work with Transaction Document Statistics

LANSA Composer provides a web-browser based interface for graphically displaying management statistics about inbound and outbound transaction document flows.

To work with Transaction Document Statistics, expand *Files* in the *Navigator* and then select *Statistics*. If the *Transaction Document Statistics* page fails to display, this is typically for one of the following reasons:

- The web server or listener is not running on the application server. If this is the case, start the web server and/or listener and retry.
- The base URL for the browser interface is not correct. Check and correct the applicable system setting if necessary. Refer to System Settings for further information.
- If the chart views do not display, it may be because the software requirements have not been met. Refer to Requirements for Chart Presentation in Transaction Document Statistics.

The *Transaction Document Statistics* page initially displays a selection of views in four panes. You can change the views by selecting from the drop-down list. You can collapse and expand the views by clicking the arrow buttons.

#### **Transaction Document Statistics**



**Note** You can open the *Transaction Document Statistics* in your web browser (outside LANSA Composer). To do this click the *Open in Web Browser* button. When opened in your web browser, you can add the console to your favourites or copy and send the URL to other users, such as operations staff.

#### **Extending or Customising Transaction Document Statistics**

The Transaction Document Statistics page as supplied offers a basic set of charts and tables relating to transaction document flows in LANSA Composer. These serve both as a starting set and as examples of what is possible.

If you wish to make more extensive use of this feature customised to your organisation's particular requirements, it is possible to develop and plug-in custom views to this page. The available views are defined in the table DXXDSHF1 in the LANSA Composer database on your LANSA Composer

IBM i or Windows server. Contact your LANSA support and service representative for more information.

### **Requirements for Chart Presentation in Transaction Document Statistics**

Four of the supplied views for the Transaction Document Statistics page are capable of presenting transaction document flow volumes in chart form.

These chart views have requirements for additional software that is not supplied or installed with LANSA Composer. These are:

- Microsoft Silverlight (version 4 recommended) must be installed on the client
- Visifire chart components run-time files must be installed on the server

The Visifire chart components are implemented in the Silverlight-based application package file *SL.Visifire.Charts.xap*. To use these chart views, you must acquire the Visifire chart components and install the *SL.Visifire.Charts.xap* in the *webserver\images* folder of your LANSA Composer server installation. For example:

- on an IBM i server, in /LANSA\_licpgmlib/webserver/images
- on a Windows server, in C:\Program Files\LANSA Composer Server\WebServer\Images

## 5. Using aXes Terminal Server with LANSA Composer

aXes is a separately licensed LANSA product that Web enables 5250 applications by dynamically transforming the 5250 data stream into a graphical interface for display in a Web browser.

LANSA Composer provides a set of activities that are designed to permit a Processing Sequence to connect to and interact with an aXes Terminal Server to navigate, populate and interrogate IBM i 5250 screens. Using the aXes Terminal Server activities, a Processing Sequence can interact directly with existing IBM i 5250 applications running on the same or a different server system. The Processing Sequence can execute activities to perform such operations as:

- connect (and logon) and disconnect the 5250 session
- interrogate the values of current screen identifiers and field values
- set the values of 5250 screen entry fields
- send function keys such as ENTER and F3
- perform multiple operations with a single activity by using aXes terminal operation scripts.

The aXes terminal server activities are individually described in:

Terminal Server Activities.

This chapter provides an overview and considerations for using these activities.

#### Also see

5.1 What can I use the aXes Terminal Server Activities for?

5.2 Requirements for using The aXes Terminal Server Activities

5.3 When to use the aXes Terminal Server Activities

- 5.4 Things you should understand about the aXes Terminal Server Activities
- 5.5 Overview of aXes Terminal Server Activities
- 5.6 Using aXes Terminal Operations Scripts
- 5.7 Processing Sequence Example

## 5.1 What can I use the aXes Terminal Server Activities for?

You can use these activities to allow your application to interact with any 5250 application, whether or not you have source code or specifications for the 5250 application. You only need to know how to operate the application as a 5250 terminal operator would.

The following describes one example scenario that might be solved with a LANSA Composer solution that employs the aXes terminal server activities.

#### To receive and enter sales orders electronically ...

When your ERP application was developed and implemented, all sales orders were received by telephone operators who entered the details at 5250 screens.

Now, many of your customers wish to submit their sales orders electronically in XML format. Unfortunately, details of the internals of the ERP application for the order entry screens are not available.

In order to satisfy the customer requirement without replacing the ERP application, you can elect to implement a LANSA Composer solution that can receive the sales order XML files, extract the details and use the aXes terminal service activities to drive the 5250 screens of the ERP order entry function.

### **5.2 Requirements for using The aXes Terminal Server Activities**

The following requirements apply to using the aXes Terminal Server activities:

- A correctly installed, licensed and configured aXes Terminal Server that is network accessible to the LANSA Composer installation is necessary to use the aXes Terminal Server activities.
- The aXes Terminal Server is a separately licensed product. Additional software license fees apply.
- The aXes Terminal Server Activities use the LANSA Integrator aXesTerminalService
- LANSA Integrator must run JVM Version 1.5 or higher in order to use the aXes Terminal Server with Composer.
- The aXesTerminalService and LANSA Composer's aXes Terminal Server activities work with aXes version 2.1 and above.

## 5.3 When to use the aXes Terminal Server Activities

In general, preferred LANSA Composer techniques for integrating with an existing IBM i application would include direct integration with:

- the application's database using Transformation Maps, and/or
- application program interfaces provided by the application using CALL\_3GL, CALL\_FUNCTION or CALL\_JAVA activities or by creating custom activities.

Using such techniques will generally provide the most resilient and the best performing solution as well as being the easiest to maintain.

However, there can be instances where the above types of integration are not possible. For example when the necessary domain knowledge is NOT available or it is NOT technically feasible to:

- directly access the application database using transformation maps.
- use application program interfaces or such interfaces do not exist.

In these circumstances, being able to drive the 5250 screens may provide a level of integration with the application that may not otherwise be possible. Where this is the case, LANSA Composer's aXes terminal server activities can provide the solution.

A solution employing LANSA Composer's aXes terminal server activities will be most suitable where:

- alternative (preferred) forms of integration (as above) have been ruled out;
- the solution will require a relatively limited amount of 5250 interaction
- the solution will involve a manageable set of individual data items (sent to or received from the 5250 screens) that can be transferred to and from the LANSA Composer processing sequence variable pool without excessive performance or maintenance burden;
- the performance of the 5250 interactions in relation to the overall solution is not critical.

If you are unsure whether your solution will meet those criteria, there are alternative techniques you may wish to consider. These can still provide the benefits of both the aXes terminal server and of LANSA Composer, but provide reduced complexity. They do, however, mean that program code must be written.

For example, you could code custom activities or functions that use LANSA

Integrator's aXesTerminalService directly to complete all the 5250 interaction associated with a known transaction in one step. Then the custom activity or function can be included in the LANSA Composer process along with associated transport, transformation and other activities and directives as required.

Alternatively, you could code directly to an aXes API for an even more efficient and direct solution.

## 5.4 Things you should understand about the aXes Terminal Server Activities

A LANSA Composer solution that uses the aXes terminal server activities differs from other LANSA Composer solutions in several important ways. It is important that you understand these differences before committing yourself to this type of solution.

For a discussion of the key differences, refer to the following:

The aXes terminal server activities are highly granular

The Processing Sequence requires greater access to application data than a typical BPI solution

The solution may be at risk of unanticipated 5250 application behavior The solution has limited eligibility for restart in the event of failure

## The aXes terminal server activities are highly granular

For any given solution, each of the LANSA Composer aXes terminal server activities will do a relatively small part of the necessary work and your Processing Sequence will need to execute a larger number of activities than would be the case with most other LANSA Composer solutions.

For example, a simple application that navigates to a data entry screen and enters some values might need to execute a sequence of activities along these lines:

- 1. TS\_CONNECT
- 2. A series of activities including TS\_SETBYPOS or TS\_SETBYNAME and/or TS\_SEND as required to navigate to the data entry screen
- 3. TS\_SETBYPOS or TS\_SETBYNAME for each of the values / fields to be filled
- 4. TS\_SEND to send the completed screen to the aXes Terminal Server
- 5. A further series of activities to navigate to a screen at which the session can be signed off in a controlled fashion
- 6. TS\_DISCONNECT

The effect of this may be reduced to some degree by using aXes terminal operations scripts (as described in 5.6 Using aXes Terminal Operations Scripts). However, this runs against the general rule that your Business Process Integration (BPI) solution should do most of its work in a smaller number of

Activities and Transformation Maps in order to optimize performance of the solution.

Therefore you should satisfy yourself that a solution employing the aXes terminal server activities will give you the performance and throughput that meets your requirements.

## The Processing Sequence requires greater access to application data than a typical BPI solution

Mostly, Processing Sequence variables hold the variable data that is used to orchestrate the process—for example, paths to transaction documents that are being processed. In the typical BPI solution, most of the transaction content (that is, the application data) is handled by and known only to the Activities and Transformation Maps that act upon it.

However, a LANSA Composer solution that uses the aXes terminal server activities must depart from this model. In such a solution, the Processing Sequence will need access to whatever data is sent to or received from the 5250 application screens.

For example, if your solution is required to fill an order entry screen, then it will need Processing Sequence variables that contain the data items (such as customer number, order date, part numbers and quantities) that must be passed as parameters to the appropriate aXes terminal server activities.

LANSA Composer provides features to facilitate this, but you should understand that this adds steps and consequent complexity and overhead to your solution. For more information refer to:

Variables

Saving, Loading and Transforming Processing Sequence Variables

## The solution may be at risk of unanticipated 5250 application behavior

Any application that seeks to interact with another application via its 5250 screens (by any means) assumes risks—for example, the 5250 application's screens may change or certain inputs provided may yield results that were not anticipated. These risks may cause unanticipated and unhandled Processing Sequence failures.

You need to understand the extent to which your proposed solution may be affected by such considerations and the degree of exposure that is acceptable to you before committing yourself to this approach.

## The solution has limited eligibility for restart in the event of failure

When a LANSA Composer Processing Sequence run ends in error, it is often possible to restart it from the point of failure—once the cause of the failure has been corrected. This is a very powerful feature of LANSA Composer.

However, a LANSA Composer solution using the aXes terminal server activities cannot be restarted if a failure occurs during the aXes terminal server session—that is, before the TS\_DISCONNECT activity is executed.

This is because LANSA Composer is unable to return the terminal server session to the state it was in at the point of the failure.

To maximize the benefit of LANSA Composer's restart capability, you should complete your aXes terminal server interactions and execute the TS\_DISCONNECT activity at the earliest opportunity. Once all active aXes

terminal sessions have been disconnected, normal restart eligibility resumes.

## 5.5 Overview of aXes Terminal Server Activities

LANSA Composer's aXes terminal server activities fall into several groups. These are:

#### **Connect and Disconnect aXes Terminal Server Sessions**

Activity ID	Description
TS_CONNECT	Connect to aXes 5250 terminal session
TS_DISCONNECT	Disconnect from aXes 5250 terminal session

#### **Interrogate aXes Terminal Session Attributes**

Activity ID	Description
TS_CAPTURE	Capture aXes 5250 terminal session screen image
TS_GET	Get aXes terminal session attributes
TS_GETFIELD	Get aXes 5250 terminal session field attributes

#### Get and Set aXes Terminal Screen Field Values and Cursor Position

Activity ID	Description
TS_GETBYNAME	Get aXes 5250 terminal session value by name
TS_GETBYPOS	Get aXes 5250 terminal session value by position
TS_SETBYNAME	Set aXes 5250 terminal session value by name
TS_SETBYPOS	Set aXes 5250 terminal session value by position
TS_SETCURSOR	Set aXes 5250 terminal session cursor position

#### Sending Data to the aXes Terminal Server

Activity ID	Description
TS_SEND	Send data to aXes 5250 terminal session
TS_SETBYNAME	Set aXes 5250 terminal session value by name
TS_SETBYPOS	Set aXes 5250 terminal session value by position
TS_SETCURSOR	Set aXes 5250 terminal session cursor position

#### **Performing Multiple Operations with Axes Terminal Operations Scripts**

You can use this activity to optimize performance and reduce complexity by performing multiple aXes terminal operations defined in a script file with a single activity execution:

Activity ID	Description
TS_EXECUTE	Execute an aXes terminal operations script

## **5.6 Using aXes Terminal Operations Scripts**

The set of aXes terminal server activities provided with LANSA Composer permit your solution to interact with your 5250 application via aXes in virtually any way necessary.

However, most of the activities perform a single operation amongst several to complete just one screen interaction. For example, the TS\_SETBYNAME or TS\_SETBYPOS activities let you set the value in a 5250 screen entry field, but only a single field for each execution.

This level of interface is the most flexible for your solution. At each and every operation, your process remains in control, and can perform different actions based on the outcome of every screen interaction.

On the other hand, if every aXes terminal operation was to be executed in this way, the size and complexity of your Processing Sequence may become excessive in a non-trivial implementation. As a consequence, the performance of your solution may be adversely affected.

To enable you to choose the appropriate balance amongst the considerations of flexibility, complexity and performance, LANSA Composer provides support for aXes terminal operations scripts that may be executed using the TS\_EXECUTE activity.

Using this support, your application can replace a series of TS\_SETBYNAME or TS\_SETBYPOS activities that populate one entry screen, for example, with a single TS\_EXECUTE activity.

You can optionally break your script into separate routines that are executed separately. For example, you may define a script routine to complete each 5250 data entry screen. For a data entry screen containing a subfile, you might define a routine to complete each subfile record or line.

Note: Your Processing Sequence may mix the use of aXes terminal operations script routines, executed using the TS\_EXECUTE activity, with direct aXes terminal server operations using other activities such as TS\_SEND and TS\_SETBYNAME. You do not have to script all of your aXes terminal server operations.

For more information refer to the following:

TS\_EXECUTE Activity

5.6.1 aXes Terminal Operations Scripts Definition

5.6.3 aXes Terminal Operations Scripts Example

## 5.6.1 aXes Terminal Operations Scripts Definition

You must provide an aXes terminal server operations script to the TS\_EXECUTE activity in the form of a comma-separated (CSV) file. To do this, you may use common tools such as a text editor or a spreadsheet program.

Following is an example of an aXes terminal operations script designed to enter the details of a received sales order. For the benefit of illustration, it is displayed in a spreadsheet program.

	A	В	С	D	E	F	G	Н	1	J	K	L
1	;axRN	axLB	axOP	axSF	axSX	axSR	axSC	axVC	axVX	axKY	axOK	axER
2	-											
3	navToOrderEntry		SEND									
4	navToOrderEntry		SET			20	7	composer request(tutord)		ENTER		
5	navToOrderEntry		SET			5	2	1		ENTER		
6	navToOrderEntry		SET			5	2	1		ENTER		
7	- 7								ACCOUNT A MONTANE OF			
8	enterHeader		SET			3	25		&EG_BCHNUM			
9	enterHeader		SET			4	25		&EG_ORDNUM			
10	enterHeader		SET			5	25		&EG_CUSNUM			
11	enterHeader		SET			6	25		&EG_ORDDAT			
12												
13	enterDetail		SET	iflda02	&THIS_LININDEX				&THIS_LINPAR			
14	enterDetail		SET	iflda0d	&THIS_LININDEX				&THIS_LINQTY			
15	enterDetail		SET	iflda16	&THIS_LININDEX				&THIS_LINPRC			
16	-											
17	quitAndSignoff		SEND							F3		
18	quitAndSignoff		SET			20	7	signoff		ENTER		

A single aXes terminal server operations script file may contain one or more script routines.

Each (non-comment) row of the comma-separated script file should contain a minimum of ten fields/columns and may contain up to thirteen. The purpose and meaning of each field/column that makes up an aXes terminal operations script is described below.

Note: Script lines beginning with a semi-colon (;) or asterisk (\*) are treated by LANSA Composer as comments and are not processed.

- Routine name of a routine to which this aXes terminal operations script line belongs. Lines for a routine must be specified contiguously in the script. The routine name is optional if you leave it blank, then the line implicitly belongs to a script routine with blank routine name.
- 2. Label name (*axLB*)associated with the line. This label may be referenced as the next action on success or failure of other lines in the same

routine/script.

3. Operation ( <i>axOP</i> )	Specifies the aXes terminal operation performed by this script line. It may be one of the following: GET (gets a field value) SET (sets a field value) SEND (sends data to the aXes terminal server) Lines containing any other value in the Operation column are ignored.
4. Field name ( <i>axSF</i> )	(GET/SET) This column may specify the name of the screen field whose value is to be set or retrieved. The name may be a user-defined name (if an aXes project name was specified on the TS_CONNECT activity) or an aXes field identifier. If the field name is not specified for a GET or SET operation, then the field row and column must be specified. NOTE: LANSA Composer supports the specification of a Processing Sequence variable name in this column. Refer to
	5.6.2 aXes Terminal Operations Script Variables for more information.
5. Field index ( <i>axSX</i> )	(GET/SET) If the field named in the previous column is contained in a subfile, this column may specify the index of the instance on the current screen of the field whose value is to be set or retrieved. The first instance on the current screen has an index of 1, irrespective of the scroll position of the subfile. If your program needs to process fields in subfiles, it must do so a screen at a time and send ROLL_UP/ROLL_DOWN using the TS_SEND activity (just like a 5250 terminal user would have to do). NOTE: LANSA Composer supports the specification of a Processing Sequence variable name in this column. Refer to 5.6.2 aXes Terminal Operations Script Variables for more information.
6. Field row ( <i>axSR</i> )	(GET/SET) This column may specify the screen row number of the field whose value is to be set or retrieved. The row and column together, if used, must refer to the first position of the required field. The value of this column is

ignored if the field name is specified. Conversely, if the field name is not specified, then the field row and column must be specified.

- 7. Field column (GET/SET) This column may specify the screen column number of the field whose value is to be set or retrieved. The row and column together, if used, must refer to the first position of the required field. The value of this column is ignored if the field name is specified. Conversely, if the field name is not specified, then the field row and column must be specified.
- 8. Field value (SET) This column may specify the literal value to be set in the specified field.(*axVC*)
- 9. Field value (GET/SET) This column may specify the name of a processing Sequence variable that will receive (GET) or provide (SET) the value for the specified screen field.

For a GET operation, the Processing Sequence variable name is required.

For a SET operation, you must specify either the Processing Sequence variable name in this column or the literal field value in the previous column.

10. Function (SET/SEND) If a value is specified in this column, the operation will send the current aXes screen data to the aXes (*axKY*)
 terminal server using the function key specified here after setting the field value for a SET operation successfully. You may use any of the function key values shown for the TS\_SETBYNAME, TS\_SETBYPOS and TS\_SEND activities. For a SET operation, if no value is specified, no send operation is performed. For a SEND operation, if no value is specified, ENTER is assumed.

# 11. Action on success (*axOK*)This column may be used to control the next script routine operation performed when this operation completes successfully. It may be one of the following:

	<ul> <li>*NEXT (the script proceeds to the next line in the same routine)</li> <li>*ERROR (the script execution ends with an error status)</li> <li>Any other non-blank value is interpreted as the Label name of another script line in the same script routine, and the script continues with the first line in the script routine that matches the specified label.</li> <li>If no value is specified, *NEXT is assumed.</li> </ul>
12. Action on error ( <i>axER</i> )	This column may be used to control the next script routine operation performed when this operation ends in error (*). It may be one of the following:
	*NEXT (the script proceeds to the next line in the same routine)
	*ERROR (the script execution ends with an error status)
	Any other non-blank value is interpreted as the Label name of another script line in the same script routine, and the script continues with the first line in the script routine that matches the specified label.
	If no value is specified, *ERROR is assumed.
	(*) An operation ends in error when, for example, a screen field is not found at the specified row or column or with the specified field name, or when a field is not input capable for a SET operation or for other cases where the requested aXes terminal server operation cannot be performed. However, such errors do not include 5250 application errors. For example, a validation error on an entered value detected by the 5250 application program is not an error for this purpose.
13. Comment ( <i>axComment</i> )	You may use this column to enter any comment text pertaining to the script, for documentary purposes. This column is reserved by the LANSA Composer aXes terminal server activities for this purpose.

## 5.6.2 aXes Terminal Operations Script Variables

You can specify a Processing Sequence variable name in selected columns of the aXes Terminal Operations script.

Specifically, the columns for which you may specify a Processing Sequence variable are:

4. Field name (*axSF*)

```
5. Field index (axSX)
```

9. Field value (field name) (*axVX*)

(this column, when used, *must* contain the name of a Processing Sequence variable).

Variable names permit you to pass and receive variable data between your Processing Sequence and the script routine by referencing variables in your Processing Sequence. The Processing Sequence variables may provide or receive screen values for SET or GET operations. They may also provide the aXes screen field name and/or index that a SET or GET operation is to reference.

To specify the name of a Processing Sequence variable, specify the variable name preceded by an ampersand (&) – for example &MYVARIABLE1.

A single script routine may reference a maximum of ninety-nine (99) different Processing Sequence variables.

Note that this feature does not support the following forms of variable references:

**Built-In variables** 

**Compound or qualified variables**—for example:

\*tradingpartner.xxx

**Indexed variables**—for example: mylist(3)

You could circumvent these limitations by assigning the desired compound or indexed variables to a simple variable name before executing the script.

## 5.6.3 aXes Terminal Operations Scripts Example

Following is an example of an aXes terminal operations script which is intended to input the details of a received sales order into the LANSA Composer Tutorial Orders application. Although it is defined in comma-separated format, it is shown here displayed in a spreadsheet program.

Script lines beginning with a semi-colon (;) or asterisk (\*) are treated by LANSA Composer as comments and are not processed.

To make this example clear, the following script routine descriptions include 5250 screen pictures that show the screens as if you were doing the order entry process manually.

This example script file is available with the LANSA Composer server system as EXAMPLE\_AXES\_SCRIPT.csv.

	A	В	С	D	E	F	G	Н	1	J	K	L
1	;axRN	axLB	axOP	axSF	axSX	axSR	axSC	axVC	axVX	axKY	axOK	axER
2	-											
3	navToOrderEntry		SEND									
4	navToOrderEntry		SET			20	7	composer request(tutord)		ENTER		
5	navToOrderEntry		SET			5	2	1		ENTER		
6	navToOrderEntry		SET			5	2	1		ENTER		
7	-											
8	enterHeader		SET			3	25		&EG_BCHNUM			
9	enterHeader		SET			4	25		&EG_ORDNUM			
10	enterHeader		SET			5	25		&EG_CUSNUM			
11	enterHeader		SET			6	25		&EG_ORDDAT			
12	-											
13	enterDetail		SET	iflda02	&THIS_LININDEX				&THIS_LINPAR			
14	enterDetail		SET	iflda0d	&THIS_LININDEX				&THIS_LINQTY			
15	enterDetail		SET	iflda16	&THIS_LININDEX				&THIS_LINPRC			
16	-											
17	quitAndSignoff		SEND							F3		
18	quitAndSignoff		SET			20	7	signoff		ENTER		

This example, contains four script routines, which are:

Script routine: navToOrderEntry

Script routine: enterHeader

Script routine: enterDetail

Script routine: quitAndSignoff

Some of these example script routines use screen row and column numbers (in columns *axSR* and *axSC* respectively) to refer to aXes screen fields. In general, we recommend using aXes screen field names assigned using the aXes developer facilities and contained in an aXes project that is named in the TS\_CONNECT activity.

## Script routine: navToOrderEntry

This script routine is for use immediately following the TS\_CONNECT activity to navigate the aXes terminal server session to the *Received Order Entry* screen, ready to enter the order details. Note the following about this script routine:

• At line 3 you can see the first operation in the script routine is *SEND*. Nothing is specified in the *axKY* column and so *ENTER* is assumed as the default function key to be sent. This line is included to allow for and handle the possibility of *Display Program Messages* screen being displayed following logon. The effect of the *SEND* is to dismiss that screen if it is displayed.



• At line 4, the script routine assumes the *System i Main Menu* is displayed, and it types the command shown on the command line to start the LANSA Composer Tutorial Orders application.

MAIN	System i Main Menu	System: LANSA01					
Select one of the following:							
1. User tasks 2. Office tasks							
4. Files, libraries, and	folders						
6. Communications							
8. Problem handling 9. Display a menu 10. Information Assistant 11. System i Access tasks	options						
90. Sign off							
Selection or command ===> <u>composer request(tutord)</u>							
F3=Exit F4=Prompt F9=Retr: F23=Set initial menu (C) COPYRIGHT IBM CORP. 1980,	ieve F12=Cancel 2007.	F13=Information Assistant	_				
MA B MW		20/0	31				

• The remaining two lines make menu selections in the LANSA Composer Tutorial Orders application necessary to navigate to the *Received Order Entry* screen.

		Received	l order entry		
Batch nur Order nur Customer Order da	nber nber number te				
Part	Line	Line			
Number	Quantity	Price			
		.00			
		.00			
	<u>0</u>	.00			
	<u>0</u>	.00			
	0	.00			
	0	.00			
	0	.00			
		.00			
		.00			
	<u>U</u>	.00			
Input or	der details a	nd press enter			Ť
F1=Help	F3=Exit F4=	Prompt F12=Ca	ncel F14=Ms	as	
				5-	
MA B		MW			03/025

#### Script routine: enterHeader

This script routine fills the order header fields of the *Received Order Entry* screen using the values contained in the processing sequence variables named in

the *axVX* column.

The Processing Sequence must populate the processing sequence variables referenced before executing this script.

## Script routine: enterDetail

This script routine fills one line of the order details portion of the *Received Order Entry* screen using the values contained in the processing sequence variables named in the *axVX* column.

The Processing Sequence variable &THIS\_LININDEX is specified in the *avSX* column to specify the index of the instance on the current screen of the field whose value is to be set—in other words, in this case, the order details line or row number.

The script routine refers to the aXes screen fields by their aXes field identifier, such as *iflda02*. More usually, you would use a user-assigned screen field name defined in the aXes project that is named on the TS\_CONNECT activity.

The Processing Sequence must populate all the processing sequence variables referenced before executing this script.



**Note:** This example assumes that there will not be more than one subfile page of order details. No scrolling is performed. Real-world applications may need to send *ROLL\_UP* using the TS\_SEND activity at appropriate points to scroll subfiles that are displayed by the 5250

## Script routine: quitAndSignoff

This script routine sends function key F3 to quit the LANSA Composer Tutorial Orders application and then signs off the terminal session.

The Processing Sequence that executes this script would usually execute the TS\_DISCONNECT activity immediately afterwards.

## 5.7 Processing Sequence Example

LANSA Composer supplies an example Processing Sequence, EXAMPLE\_TS01 that uses the example aXes terminal operations script described in the 5.6.3 aXes Terminal Operations Scripts Example to populate data in the LANSA Composer Tutorial Orders application.

Part of the definition of this Processing Sequence follows. You can run this Processing Sequence if you have all the necessary components installed and configured, including, of course, the aXes Terminal Server.



## 6. Deploying Solutions for LANSA Composer

A common implementation scenario for LANSA Composer involves using two installations:

- 1. One for designing and testing LANSA Composer Solutions ("development")
- 2. One for running LANSA Composer Solutions in a live environment ("production")

Separating the 'development' and 'production' systems has many obvious advantages and is recommended for most cases. When you maintain separate LANSA Composer installations, you will need to deploy LANSA Composer solutions between them.

Following are some considerations for deploying your systems and the features LANSA Composer provides to support them. For more information, review the following:

- 6.1 Deployment Considerations
- 6.2 Work with Export Lists
- 6.3 Export Definitions from LANSA Composer
- 6.4 Import Definitions to LANSA Composer

## **6.1 Deployment Considerations**

When you maintain separate systems for development and production work, you must understand the steps necessary to successfully and completely deploy a LANSA Composer solution from one system to the other.

Composer provides export and import features to support your deployment needs. The export and import features are capable of largely automating most of the work of deploying LANSA Composer solutions, but there are some steps that you must perform yourself and some considerations you should take account of.

These considerations apply to:

Activities

Activity Groups

**Trading Partners** 

**Transformation Maps** 

Processing Sequences

Transport and Other Configurations

Security Credentials for Transport and Other Configurations

Code Values

System Properties

System Settings

### Activities

If you define your own custom Activities, then you will need to deploy them to all systems on which you wish to use them.

There are two parts to a complete and functioning Activity:

- 1. **The Activity definition** you can use LANSA Composer's export and import support to deploy the Activity definitions.
- 2. **The Activity processor** this is the executable code (a LANSA reuseable part) that implements the Activity. It is your responsibility to deploy the Activity processor as required. You might wish to use LANSA's export and import support or the Visual LANSA Deployment Tool to accomplish this. Refer to your LANSA documentation for information about deploying LANSA applications.

You should not normally need to export or import the definitions of Activities supplied with LANSA Composer other than as instructed during an installation or upgrade of LANSA Composer. You should not change the definitions of these Activities.

## **Activity Groups**

When you export Activity definitions, LANSA Composer exports the Activity Groups to which the Activity is assigned on the source system.

When you import the Activity definitions to the target system, LANSA Composer will assign the Activity to the same Activity Groups, *but only if they already exist on the target system*. LANSA Composer will not create the Activity Group on the target system. You must deploy or define the Activity Groups to the target system prior to importing the Activity definitions.

Note that failing to assign the Activity Groups on the target system will not affect the functioning of the Activity or of Processing Sequences that use it. Activity Groups are simply a means of categorizing the Activities for quick access while maintaining Activity definitions or when editing a processing sequence.

## **Trading Partners**

You can deploy trading partners between LANSA Composer installations using the export and import features, in the same way as other definitions.

If you are using installation-defined properties for Trading Partners, then the values defined for those properties for each Trading Partner will be included in the export. When you import the Trading Partner definitions to the target system, LANSA Composer will import the values for the installation-defined Trading Partner properties, *but only if they already exist on the target system*. You must deploy or define the Trading Partner installation-defined properties to the target system prior to importing the Trading Partner definitions. The same applies to Trading Partner groups.

When you export trading partner definitions, the export includes the definition of which configurations, transformation maps and directories are linked to the trading partner. However the actual configurations and transformation maps are not automatically included - it is your responsibility to ensure the configurations and transformation maps exist on the target system. If they do, then the trading partner import will restore the links.

With regard to the linked directories, in particular, you may wish to structure the
directory paths such that they are the same on each installation. If you don't, you may have to manually alter this aspect of trading partner definitions on the target system after importing them.

# **Transformation Maps**

The complete definition of a Transformation map in LANSA Composer consists of:

- 1. Those parts held in the LANSA Composer database. For a map that has been successfully prepared, this includes:
  - Basic identifying information such as the name and description for the map;
  - Information about the parameters that must be supplied to execute the map;
  - Catalogued information about the map including information identifying the major components used in the map definition;
  - Catalogued copies of the map definition and implementation files as at the last successful prepare.

When you export Transformation maps, all this information is included in the export, including the map definition and implementation files (subject to the options chosen in *Export and Import Options*).

2. The files held in the file system that are associated with the Transformation Map such as the map definition (.mfd) and map executable (.jar) files.

When you export Transformation maps that have been successfully prepared with LANSA Composer version 4.0 or later, LANSA Composer exports the catalogued versions of the files (from the LANSA Composer database) in preference to the files held in the file system. If the catalogued files do not match those held in the file system, LANSA Composer will issue a warning in the export log, but the catalogued versions are exported nonetheless.

If the catalogued versions of the files are not available (for example, if the map has not been prepared or if the map was prepared with an earlier version of LANSA Composer), LANSA Composer exports the versions of the files held in the file system.

Some other points to note about deploying Transformation maps:

• The Transformation Map definition (.mfd file) is not required at run-time. It is only required to display or revise the Transformation Map using the

mapping tool. Whether to copy and deploy these definitions is a matter of source and version control rather than a deployment requirement.

- LANSA Composer's export and import support can deploy the map executable (.jar) files as described above and according to the options chosen in *Export and Import Options*. Normally you should deploy these files, since the affected Transformation Maps cannot be executed without them.
- When importing a map for which an MFT (Flextext configuration) file was catalogued and exported, the MFT file is catalogued in LANSA Composer's file store database on the importing system, but it is NOT written to the file system (UNLIKE the MFD and JAR files). This is deliberately so, since such files may be (often will be) shared between more than one map. Also, because there is no LANSA Composer-mandated location for such files (it is specified in the map definition).

Note: Including the Transformation Map executable files (the .*jar* files) in an export will very considerably increase the size of the export file. For easiest handling and to avoid exceeding system limits, you may need to modularize or separate the export of individual Transformation Maps or groups of Transformation Maps to allow for this.

### **Processing Sequences**

When you export Processing Sequence definitions, the exported definition does not include the version history for the Processing Sequence. Only the current definition is exported.

# **Transport and Other Configurations**

The transport and other configurations feature of LANSA Composer provides a separation between the configuration information (which might vary from one system to another) and the Processing Sequences that use them.

This means, for example, that when you deploy a Processing Sequence from one system to another, while the Processing Sequence uses a set of named configurations, the actual definitions of the configurations can be different on the 'development' system to the 'production' system. Thus the same Processing Sequence can be executed on both systems using the transport and database configuration details appropriate to that system.

If you use Configurations in this way, then you probably will not wish to deploy them between systems, other than when you initially create them. If you do deploy them, you will need to change their definitions after importing them to the target system.

For Configurations that are dependent on external files or objects (such as the body text file for an SMTP Mail Details Configuration), it is your responsibility to ensure that the external files or objects are deployed to or accessible from the target system.

# Security Credentials for Transport and Other Configurations

LANSA Composer does *not* export or import passwords associated with security credentials contained in transport and other configurations. This means that you must manually change the configuration on the target system after importing it the first time, to specify the password (and, probably, the user) that applies on that system.

User names, where specified, are exported. However, it is likely in many cases that the user names will also have to be changed on the target system.

LANSA Composer will *not*, by default, replace the security credentials on the importing system when a configuration being imported already exists. This means you can import the configurations again without having to re-enter the user and password details. The *Export and Import Options* window allows you to override this behavior, if you wish, such that the user names are imported and the existing password data is invalidated (passwords will have to be re-entered).

# **Code Values**

LANSA Composer's export and import support can deploy code values. When you export a group of code values (for example, Activity Groups), all the code values in that group are exported. When you import them to the target system, they add to or update any code values already present. Code values defined on the target system that are not included in the export are not removed.

# **System Properties**

You can deploy System Properties using LANSA Composer's export and import features.

When you import System Properties, by default LANSA Composer will *not* replace the *value* of the System Property if it already exists on the importing system. The *Export and Import Options* window allows you to override this behavior, if you wish

When you deploy System Properties that are evaluated and/or set by means of a call to a user-specified function, it is your responsibility to deploy the

### required function(s) to the target LANSA Composer system.

## **System Settings**

LANSA Composer does not export or import System Settings. It is your responsibility to ensure that the System Settings are defined on the target system with values appropriate for that environment. You normally only need to do this once.

# 6.2 Work with Export Lists

LANSA Composer supports creating and using Export Lists consisting of saved lists of any LANSA Composer definitions eligible to be exported. By defining an Export List you can reliably repeat the same deployment operation over time or for different targets. You can create, maintain, and delete any number of Export Lists according to your application's needs.

An Export List can also include other Export Lists. When such a list is exported, LANSA Composer will explode the list (disregarding any circular references) and export the super-set of definitions contained in all the referenced export lists.

To work with Export Lists, expand *Deployment* in the *Navigator* and click *Export lists*. To find out how to locate and select Export Lists to work with, refer to:

#### Locating and selecting items in the Instance Lists

For information on common tasks associated with Export Lists (such as creating, copying, deleting and printing definitions) refer to:

Working with definition items

For information on tasks associated specifically with Export Lists, refer to the following headings:

Select Items to be Included in an Export List

#### Select Items to be Included in an Export List

To review or change the LANSA Composer definitions included in an Export list, select the required item in the instance list and then click the *Items* tab. The window now displays two further tabs:

- *Items*: Expand the nodes to review the available definitions. Select definitions you wish to include in the Export list by clicking the checkbox to the left of the item so that there is a tick in the box. To deselect an item, click the checkbox again so that the tick mark is removed. You can use the *Select All* and *Unselect All* buttons and the checkboxes next to the nodes to select or unselect multiple items at once.
- *Include*: Select other Export Lists that you wish to include in this Export List by clicking the checkbox to the left of the item so that there is a tick in the box. To deselect an item, click the checkbox again so that the tick mark is removed. You can use the *Select All* and *Unselect All* buttons to select or unselect multiple items at once.

Remember to click *Save* before switching elsewhere in LANSA Composer.

### **6.3 Export Definitions from LANSA Composer**

To export LANSA Composer definitions, perform the following steps:

- 1. Expand *Deployment* in the *Navigator* and click *Export*.
- 2. If you wish to export LANSA Composer definitions specified in an Export list, select the Export list name from the drop-down list. When you select an Export list, the item selections are changed according to the contents of the Export list. You can still change the selections before performing the export, if desired.
- 3. If you do not wish to use an Export list, select *(none)* in the Export list drop-down.
- 4. On the *Items* and/or *Include* tabs, expand the nodes to review the available definitions.
  - a. Select definitions you wish to export by clicking the checkbox to the left of the item so that there is a tick in the box. To deselect an item, click the checkbox again so that the tick mark is removed.
  - b. You can use the *Select All* and *Unselect All* buttons and the checkboxes next to the nodes to select or unselect multiple items at once.
  - c. Whenever you wish to, you can click *Save list* to save the current selections as an Export list for re-use in the future.

🗸 Export	
Export list:	
(none)	×.
Export <u>R</u> efresh	Select <u>All</u> <u>Unselect All</u> <u>Options</u> <u>Save list</u>
Name	Description
Activities	
🛨 📃 Trading partners	
+	
Processing sequence	
EXAMPLE_AATEST1	Check LANSA Composer installation
EXAMPLE_AATEST2	Check LANSA Integrator functionality
EXAMPLE_ACTPARM	Shows ways to provide activity parameter values
EXAMPLE_CSV01	Example of processing CSV
EXAMPLE_LOG01	Logging Example
EXAMPLE_LOOP01	Example of processing multiple documents.
EXAMPLE_LOOP02	Example of processing multiple trading partners.
EXAMPLE SUSP01	Suspend Example
EXAMPLE SWIT01	Terminate, loop and conditional example
FSTUTSEQ	Fred Smith LANSA Composer tutorial

The *Include* tab allows you to select other Export Lists that you wish to include in the export, in addition to any definitions selected on the *Items* tab.

- 5. If necessary, click the *Options* button to change options affecting the export in the *Export and Import Options* window. Click *OK* to save your changes and close the *Export and Import Options* window.
- 6. When you have completed your selections, click the *Export* button. The *Export To* window is displayed.
- 7. Enter the file name for the exported definitions in the *File name* box.

If necessary, navigate to the folder in which you want to save the export file and click the *Open* button. LANSA Composer exports the definitions to the file and location specified.

When the export is completed, LANSA Composer shows the *Export Log* which provides a detailed record of the export operation including any warnings or errors that may have been generated.

You can display or print logs for any or all completed export operations at any time by clicking the *History* tab.

From the *History* tab, you can also print an *Export Manifest* that lists all exported items along with their last-changed dates, times and users. This can be used to compare with last-changed details shown in LANSA Composer for selected items on their *Audit* tab.

NOTE: LANSA Composer does not automatically resolve dependencies between items when exporting their definitions. For example, it does not automatically export Transformation Map definitions used by a Processing Sequence. It is your responsibility to ensure that all required definitions are exported.

Refer to 6.1 Deployment Considerations for other tasks you may have to complete to successfully deploy a LANSA Composer solution.

# **6.4 Import Definitions to LANSA Composer**

To import definitions, you must:

- have LANSA Composer installed on the target PC
- an export file previously created by the *Export* feature of LANSA Composer. Refer to 6.3 Export Definitions from LANSA Composer for information on creating an export file.

To import LANSA Composer definitions, perform the following steps:

- 1. Expand *Deployment* in the *Navigator* and click *Import*.
- 2. Press the *Load* button to display the *Import From* window.
- 3. Locate and select the file containing the definitions you wish to import and click *Open*. LANSA Composer reads the exported definitions and then shows the available definitions in the *Import* window.

🍇 Import				
Load Import	Refresh Select All Unselect All	Options		
C:\Composer\my_export.xml				
Name	Text	Changed by User	Change date	Change Time
표 🔲 🛕 Transform				
🖃 📝 🛕 Processing seque				
Image:	Extended order receive processing	QOTHPRDOWN	2008/12/01	14:18:15
	Processing sequence 'DEM_ORDER_EXT' already exist			
Image: Second	Order receive processing	QOTHPRDOWN	2008/12/01	14:18:16
	Processing sequence 'DEM_ORDER_RCV' already exist			

- 4. Expand the nodes to review the available definitions. Take note of any warning messages that are displayed.
- 5. Select definitions you wish to import by clicking the checkbox to the left of the item so that there is a tick in the box. To deselect an item, click the checkbox again so that the tick mark is removed.

You can use the *Select All* and *Unselect All* buttons and the checkboxes next to the nodes to select or unselect multiple items at once.

- 6. If necessary, click the *Options* button to change options affecting the import in the *Export and Import Options* window. Click *OK* to save your changes and close the *Export and Import Options* window.
- 7. When you have completed your selections, press the *Import* button to import the selected definitions.

When the import is completed, LANSA Composer shows the *Import Log* which provides a detailed record of the import operation including any warnings or errors that may have been generated.

You can display or print logs for any or all completed import operations at any time by clicking the *History* tab.

From the *History* tab, you can also print an *Import Manifest* that lists all imported items along with their last-changed dates, times and users. This can be used to compare with last-changed details shown in LANSA Composer for selected items on their *Audit* tab.

Refer to 6.1 Deployment Considerations for other tasks you may have to complete to successfully deploy a LANSA Composer solution.

For instructions on how to restore shipped objects, please review:

6.4.1 Restore Supplied Definitions

# **6.4.1 Restore Supplied Definitions**

LANSA Composer keeps a 'master store' of supplied definitions with each standard installation of the LANSA Composer server. These can be found in the *Imports* directory of the Composer server shared directory. For a default installation, this will typically be:

/LANSA\_Composer\_licpgmlib/lic/Imports. Otherwise, it will be the IFS file folder or Data Directory as specified during the installation.

To restore shipped objects, please use the *Import* facility in LANSA Composer. The objects that can be imported are as follows:

DX\_MASTER\_AT.dxexport for Activities

DX\_MASTER\_CFG.dxexport for Configurations

DX\_MASTER\_DOC.dxexport for Document types and Document standards

DX\_MASTER\_EV.dxexport for Event handlers

DX\_MASTER\_MP.dxexport for Transformation maps

DX\_MASTER\_P1.dxexport for Processing Sequence

DX\_MASTER\_TP.dxexport for Trading partners

DX\_MASTER\_XL.dxexport for Export List.

In addition, the following may also be imported. However, importing these may override existing definitions.

DX\_MASTER\_CH.dxexport for Code tables

DX\_MASTER\_SY.dxexport for System Properties.

To learn more about importing definitions, please refer to 6.4 Import Definitions to LANSA Composer.

# 7. Operations

LANSA Composer provides a web-browser based interface for operational monitoring and control. It is accessible within the LANSA Composer environment by expanding *Operations* in the *Navigator*. The following options are available:

7.1 Console - for submitting and monitoring your Processing Sequence runs for errors or other conditions that may require operator intervention.

**7.2 Java Service Manager Console** - for performing administration tasks on the java service manager (JSM) instance associated with this installation of LANSA Composer

# 7.1 Console

The console allows you to submit and restart Processing Sequence runs, monitor your Processing Sequence runs for errors or other conditions that may require operator intervention and to review the Processing Sequence logs for individual runs.

To access the console, expand *Operations* in the *Navigator* and select *Console*. If the console fails to display, this is typically for one of the following reasons:

- The web server or listener is not running on the application server. If this is the case, start the web server and/or listener and retry.
- The base URL for the browser interface is not correct. Check and correct the applicable system setting if necessary. Refer to System Settings for further information.

The console initially displays the 7.1.1 Run History page. From the Run History page, you can choose to display the 7.1.2 Processing Sequence Log for an individual Processing Sequence run.

You can also display a list of active Processing Sequences and submit runs for them using the 7.1.3 Processing Sequences page.

**Note:** You can open the console in your web browser (outside LANSA Composer). To do this click the *Open in Web Browser* button. When opened in your web browser, you can add the console to your favorites or copy and send the URL to other users, such as operations staff.

# 7.1.1 Run History

By default, the console initially displays the Run History page. You can display or return to the Run History page by clicking the *Run History* button at the top of the page.

The Run History page shows a list of active and completed Processing Sequence runs in descending order of their start date and time. The most recent runs are shown first.

You can **filter** the list by entering or selecting values in the boxes immediately under the *Processing Sequence ID*, *Start Date Time* and *Status* column headings and pressing the *Refresh* button.

Refresh		▲ Later	Earlier 🕨				
Name		Run Number	Start Date / Time		End Date/Time	Sta	tus
DEM_ORDER_RCV			2008-12-04 📖			All	•
DEM_ORDER_RCV	Q	000000000000091	2008-12-04 14:14:13	200	08-12-04 14:14:41	0	ок
DEM_ORDER_RCV	Q	000000000000000000000000000000000000000	2008-12-04 14:12:02	200	08-12-04 14:12:28	0	ок
DEM_ORDER_RCV	Q	000000000000089	2008-12-04 14:11:10	200	08-12-04 14:11:15	0	ER
DEM_ORDER_RCV	Q	00000000000088	2008-12-04 13:42:22	200	08-12-04 13:44:22	0	ок
DEM_ORDER_RCV	Q	000000000000087	2008-12-04 13:42:11	200	08-12-04 13:44:17	0	ок
DEM_ORDER_RCV	Q	000000000000086	2008-12-04 13:42:07	200	08-12-04 13:44:09	0	ок

The Run History is shown one page at a time. Click the *Later* or *Earlier* buttons to see later or earlier Processing Sequence run history.

To see the Processing Sequence log for a run shown in the list, click the icon or the run number shown in the *Run Number* column. Refer to 7.1.2 Processing Sequence Log for more information.

# 7.1.2 Processing Sequence Log

The Processing Sequence Log shows the status and log for an individual run of a Processing Sequence.

Back to Run History		Restart				
Back to Run History Processing sequence Description Run number Start date / time End date / time Execution result code Show Summary  Processing sequence Parameter value Parameter value Assignment comp III Starting activitit III Open connect Load service F	DEM_OR Drder re- 20000000 2008-12- 200-	Restart DER_RCV ceive processing 00000089 04 14:11:10 04 14:11:15 rted INBOUND IP Get files from remote	: host	Message ID Message type Message text Log date time Activity ID Activity description	DXS0113 *ESCAPE ERROR - Connection refused: connect 2008-12-04 14:11:12 FTP_INBOUND FTP Get files from remote host	
FTP Connect     FTP Connect     JSM *SERVIC     JSM *TRANSP     Event notifical     SActivity endec     Finished activity f     Processing sequen	t to host to host E trace of ORT trac ORT trac tion hand tin error TP_INB nce end	data is available ce data is available dled r OUND <b>led</b>				

If there are detail log entries available, you can toggle between a detailed and summary view by using the *Show All* and *Show Summary* buttons.

You can click on a log entry in the tree on the left to see the message details in the panel on the right.

If the Processing Sequence run ended in error and is eligible to be restarted, you can restart it by clicking the *Restart* button. The *Restart Processing Sequence Run* page is displayed at which you can confirm and submit the run.

D DEM_ORDER_RCV
escription Order receive processing
0000000000089
2008-12-04 14:11:10
2008-12-04 14:11:15
ER
Cancel

### 7.1.3 Processing Sequences

You can display the Processing Sequences page by clicking the *Processing Sequences* button at the top of each page.

The Processing Sequences page shows a list of active Processing Sequence definitions.

You can **filter** the list by full or partial Processing Sequence name by entering a value in the box immediately under the *Name* column heading and pressing the *Refresh* button.

Refresh			•	Previous	Next	•
Run	History	Na	ame	Description	I.	
		DEM				
	Q	DEM_OR	DER_EXT	Extended of	rder receive pro	ocessing
	Q	DEM_OR	DER_RCV	Order recei	ve processing	
	Q	DEM_OR	DER_RCV2	Order recei	ve processing	

The Processing Sequences are shown one page at a time. Click the *Previous* or *Next* buttons to see Processing Sequences before or after those presently shown.

To display the 7.1.1 Run History of a selected Processing Sequence, click the icon shown in the *History* column adjacent to the required Processing Sequence.

To submit a run of a selected Processing Sequence, click the icon shown in the *Run* column adjacent to the required Processing Sequence. The *Run Processing Sequence* page is displayed at which you can enter parameter values (if applicable) and submit the run.

Processing sequence D Description 0	EM_ORDER_RCV rder receive processing	
Type parameter values	for this run:	
Customer's trading partne	r id ATLAS	
Customer's remote directo	ry //neworders'	
Run	Cancel	

### 7.2 Java Service Manager Console

The Java Service Manager Console allows you to perform administration tasks on the java service manager (JSM) instance associated with this installation of LANSA Composer. When used from within LANSA Composer, this includes the ability to stop and restart the JSM - an operation that is necessary in order to use newly prepared or deployed Transformation Maps.

To access the Java Service Manager Console, expand *Operations* in the *Navigator* and select *Java Service Manager Console*. If the Java Service Manager Console fails to display, this is typically for one of the following reasons:

- The JSM instance is not running. If this is the case, start the JSM instance and retry.
- The URL for the console for JSM instance is not correct. Check and correct the applicable system setting if necessary. Refer to System Settings for further information.

**Note:** You can open the Java Service Manager Console in your web browser (outside LANSA Composer). To do this click the *Open in Web Browser* button. When opened in your web browser, you can add the console to your favorites or copy and send the URL to other users, such as operations staff.

Refresh Open in Web Brov	vser	Restart JSM
Java Service Manage	er Conso	le
	Passive	Active
Active service count	Execute	
Refresh resources	Execute	
System information	Execute	
Perform garbage collection	Execute	
Controlled shutdown		Execute
Close console server		Execute

The Restart JSM button stops and restarts the java service manager (JSM). It

may sometimes be necessary to stop and restart the JSM if new LANSA Integrator solutions or other new or revised Java code is deployed to LANSA Integrator or if certain changes are made to LANSA Integrator configuration. (Note it is no longer necessary to stop and restart the JSM when you have prepared or deployed new or revised Transformation Maps).

When you click the *Restart JSM* button, a message box asks you to confirm the request before proceeding. This is because stopping and restarting the JSM may interfere with other applications, especially in a production environment. If you are sure you wish to proceed, click Yes.

**Note:** The *Restart JSM* button and adjacent buttons are not part of the web-browser interface for the Java Service Manager Console and are not available when you open the Java Service Manager Console in a web browser.

For more information about the Java Service Manager Console, refer to the Java Service Manager Console in the *LANSA Integrator Guide*.

# 8. Administration and Housekeeping

LANSA Composer provides administration and housekeeping functions for you to manage system and code values and perform routine maintenance.

To access the administration and housekeeping functions, expand *Administration and Housekeeping* in the *Navigator* and then select the required function.

The following administration and housekeeping functions are available:

- 8.1 Audit Trail
- 8.2 System Settings
- **8.3 System Properties**
- 8.4 Code Maintenance
- 8.5 Event Maintenance
- 8.6 Database Housekeeping
- 8.7 User Access Configuration

# 8.1 Audit Trail

*Audit trail* is available under *Administration and Housekeeping* to permit interrogation and viewing the events recorded in the audit trail.

LANSA Composer maintains an audit trail of significant (usually modifying) events to most definitions including:

- Activities
- Trading Partners
- Transformation Maps
- Processing Sequences
- All configuration types
- Export Lists
- Document types and standards
- Event definitions

Events that are recorded include:

- Create, change, delete
- Opened for editing
- Saved (Processing Sequences)
- Prepared (Transformation Maps)
- Archived version restored, deleted or purged (Processing sequences and Transformation Maps)
- Exported, Imported

Information that is recorded for each event includes:

- The identity of item affected
- Timestamp
- Related log, where applicable (export, import)
- Job name, user and number
- Computer name

Note that the audit records do NOT record what actually changed in the item definition, only that the event occurred.

The *Audit trail* option provides several tabs that allow you to select and view

the audit events in various ways.

denniuon type   By addit even	Dy date [ by user   by comp	outer				🚽 🔄 Refre
Audit event	Timestamp	Name	Description	Log	User name	
🗄 🛗 Today						D <u>e</u> tails.
Archived version res	2013-08-22 1:16:00 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN =	
Archived version res	2013-08-22 1:15:54 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	Show Loo
Changed (Details)	2013-08-22 1:15:39 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Saved after editing	2013-08-22 1:15:25 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Opened for editing	2013-08-22 1:15:11 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Changed (Attachme	2013-08-22 1:15:04 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Changed (Notes)	2013-08-22 1:14:29 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Saved after editing	2013-08-22 1:14:06 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Opened for editing	2013-08-22 1:13:51 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Created as copy of	2013-08-22 1:13:47 PM	MY_SQL01	Example of using the SQL		QOTHPRDOWN	
Export using export I	2013-08-22 12:36:10 PM	DX_MASTER_EG	Shipped examples	00000000000075	QOTHPRDOWN	
Exported	2013-08-22 12:36:10 PM	EXAMPLE_TP	Example trading partner d	00000000000075	QOTHPRDOWN	
Exported	2013-08-22 12:36:09 PM	EXAMPLE SMS	Example SMS configuration	000000000000075	QOTHPRDOWN	

When working with specific definition items (for example, with a specific Processing Sequence), you can use the *Audit* command tab to display the audit events just for that item.

# 8.2 System Settings

The system settings enable you to set values which are used throughout the LANSA Composer application.

To reach the system settings, expand *Administration and Housekeeping* in the *Navigator* and then select the required function.

The system settings are grouped into these logical sections to make it easier to locate and work with the values:

- 8.2.1 Logging
- 8.2.2 Server File Locations
- 8.2.3 File Locations Relative to Client
- 8.2.4 Browser
- 8.2.5 SMTP Mail Details Defaults
- 8.2.6 HTTP Inbound Processing
- 8.2.7 Transaction Document Processing
- 8.2.8 Default Configuration
- 8.2.9 Other Settings

Each of the system settings is explained and an example of a possible value is included. The values defined for your implementation of Composer will not necessarily be the same as the examples.

Several of the system settings contain directory paths. The path name directory delimiters should be appropriate for the operating environment. MS Windows delimiters are backward slashes (\) and IBM iare forward slashes (/). Paths for server directories should be appropriate for your server. Paths for client directories should be MS Windows format.

If you use different development and production environments (the recommended approach), you should review and reset, as necessary, these settings when you deploy your solutions to the production environment.

**Note:** The names of system settings that you have changed but not yet saved are shown in blue in the system settings tree on the left of the tree. Remember to press the *Save* button to save your changes.

Systems Settings also provides a System information tab that provides information about the LANSA run-time environment in which LANSA Composer is running on your LANSA Composer server, including LANSA

# listener and JSM port numbers:

Details (1) System information		
Listener port:	05592	
Web instance port:	05593	
LANSA Integrator (JSM) port:	05594	
LANSA Integrator (JSM) admin port:	05595	

# 8.2.1 Logging

This section contains values that can be used to limit the logging messages which are captured during the execution of a Processing Sequence. Typically all messages would be reported while you are designing and testing your LANSA Integrator solutions, but only summary and error level messages would be captured in a production environment.

Note that the settings in this section do not affect the logging of system tasks such as Export, Import and Database Housekeeping. These tasks are always logged at a predetermined level.

🔯 System settings	
<ul> <li>System settings</li> <li>System Settings</li> <li>Logging</li> <li>Logging level</li> <li>LANSA Integrator tracing</li> <li>Server file locations</li> <li>Tutorial files</li> <li>Default trading partner linked directory</li> <li>File locations relative to client</li> <li>Server network path</li> <li>Map generate and compile</li> <li>LNSA Exercise (2014) indexes</li> </ul>	Select: Maximum
<ul> <li>Enclose Antegration (comparison for the processing sequence versions zions) in statuce</li> <li>Browser</li> <li>Browser</li> <li>Base URL for browser interface</li> <li>URL for LANSA Integrator (JSM) console</li> <li>SMTP mail details defaults</li> <li>From e-mail address</li> <li>From e-mail display name</li> <li>From e-mail display name</li> <li>HTTP inbound processing</li> <li>HTTP inbound processing</li> <li>HTTP inbound file name prefix</li> <li>Obfault Configuration</li> <li>SMTP server configuration</li> <li>SMS configuration</li> <li>SMS configuration</li> <li>Cother settings</li> <li>Archive processing sequence versions</li> <li>Zip archive file name prefix</li> </ul>	Hint
POP3 save sub-directory name prefix From e-mail address for event notifications Job description for Log Purge job	This setting determines the level of messages that are included in the processing sequence log.

### Logging level

This setting determines the level of messages that are included in the Processing Sequence log. You can choose one of the following from the drop-down list:

**Automatic** For a successful processing sequence run, the log will record completion and some other messages for most activities and

transformation maps. However detailed logging (equivalent to the Maximum logging level) is held in memory in a rolling cache. If an error occurs, the contents of the cache are flushed to the log, providing detailed logging where it is needed - preceding and at the location of the error.

- **Minimum** A minimal level of messages are included in the log. For a successful processing sequence run, the log will simply record the start and end of the processing sequence.
- **Normal** Completion messages for most activities and transformation maps are included in the log.
- **Maximum** Extensive logging records the progress of the processing sequence and each activity and transformation map..

The suggested values are Automatic or Maximum during design and testing and Automatic or Normal in a production environment. Automatic is a good choice that balances the performance impact of extensive logging against the necessity to log sufficient information to enable problem diagnosis.

### LANSA Integrator Tracing

When enabled, this sets LANSA Integrator tracing on when an Integrator service is started by an Activity supplied with LANSA Composer.

This setting overrides the corresponding trace properties specified in the LANSA Integrator manager.properties file for supplied Activities executed through a LANSA Composer Processing Sequence.



For supplied activities used at the maximum logging level, the LANSA Integrator trace information will be available in the processing sequence log.

If you are developing your own custom Activity processor that uses a

LANSA Integrator service, you can respect this value by checking the IntegratorTrace property of the ancestor class and setting the TRACE keyword of the SERVICE\_LOAD command accordingly. Alternatively, if you load your service by means of a call to the ExecuteJSMLoad method provided by the DXACTBAS1 ancestor class, the tracing will be enabled automatically for you.

### 8.2.2 Server File Locations

The values in this section specify directory paths used by LANSA Composer as defaults, or for locating or storing files.

The paths specified in this section are used by software running on the LANSA Composer server and must be specified relative to and in the form appropriate for the server system.

LANSA Composer does not validate the paths specified here. You must make sure that the paths are correct for your installation. You can use the *Browse* button to browse the server file system for a directory path.



### **Tutorial files**

# Example: /LANSA\_Composer\_licpgmlib/lic/Tutorial

This value should contain the location on your server where the tutorial data has been placed. Refer to *Objects Supplied for the Tutorials* in the Tutorial installation for IBM i Server for the files that this directory should contain.

If you intend to complete the tutorials, please check that the value entered here is correct.

### **Default trading partner linked directory**

### Example: /LANSA\_Composer\_licpgmlib/lic/

This value is used when a linked directory is created for a Trading Partner. This should be a directory on your server. This value is set as the default and may be changed or extended in the Linked directory maintenance. For example, the

base directory could be /LANSA\_Composer\_licpgmlib/lic/ and within that there could be sub-directories for each trading partner such as /LANSA\_Composer\_licpgmlib/lic/TP1/inbound.

### 8.2.3 File Locations Relative to Client

The values in this section specify directory paths used by LANSA Composer as defaults, or for locating or storing files.

The paths specified in this section are used by software running on the LANSA Composer client and must be specified relative to and in the form appropriate for the client system.

For several of the paths in this section, you can specify two separate values:

1. Default value

This is the default value for all client installations and the value is saved on the server system. If all your clients access the server file system in the same way (for example, using a fixed mapped drive letter or UNC path), then you can set this value once and all clients will use the same value (unless overridden on that client). This is the recommended approach.

2. Override value for this client

This value overrides the default value for this client installation. The override value (if used) is stored only on the client system and can be different on each client system. If there is no client override value specified on the current client, the default value is shown.

LANSA Composer performs delayed validation on the paths specified here and will highlight in red paths that could not accessed. However, you can save the values entered whether LANSA Composer could access them or not. You must make sure that the paths are correct for your installation.

You can use the *Browse* button to browse the file system for a directory path.



#### Home paths

The *home path relative to the client* defines a path that the LANSA Composer client can use to access a common "home" location on the LANSA Composer server. This value is used in the LANSA Composer client software for a number of operations, including for preparing Transformation Maps. (It is not used at run-time on the LANSA Composer server, however.)

The *home path relative to the server* should define the server path to the same location. The LANSA Composer client software uses this to *convert* paths between client-relative and server-relative form, where necessary. This permits, for example, direct opening from the client application of certain files whose paths are server-based, such as the body text file for an SMTP mail details configuration.

Example1 (for LANSA Composer on an IBM i server):

*Home path relative to the client:* \\qiseries\ifs

Home path relative to the server: /

Example2 (for LANSA Composer on an IBM i server):

*Home path relative to the client:* 

\\qiseries\ifs\LANSA\_Composer\_licpgmlib\lic

Home path relative to the server: / LANSA\_Composer\_licpgmlib/lic

Example3 (for LANSA Composer on a Windows server):

*Home path relative to the client:* \\server01\LCShare\Data

*Home path relative to the server*: C:\Program Files\LANSA Composer Server\Data

### **Map definitions**

Example: \Composer\Map

LANSA Composer combines this value with the home path to determine the location of the Transformation Map definitions. The resulting path must exist on your application server.

#### Map generate and compile

Example: C:\Temp

This value specifies the location where intermediate build files (such as Java source and object files) will be held during the process of preparing a transformation map. The default setting is to use the Windows temporary directory for the user, but you can explicitly specify a location by clicking the second radio button. For best performance while preparing transformation maps, it is recommended that a location on the local (client) file system is used.

em Settings	Map generate and compile	
Logging Logging level LANSA Integrator tracing	Use the Windows temporary directory for the us	er
Server file locations	C: \Users\Hugh.SYD\AppData\Local\Temp\	Browse
SEF files		Reset Value
Default trading partner linked directory		a a construction of the second se
File locations relative to client		
Home path	Hint	
Map definitions Map generate and compile	This value specifies the location where intermediate object files) will be held during the process of prepar are not used or required after the transformation m	build files (such as java source ar ing a transformation map. The file an bas been prepared

The files in this directory are only used during the Prepare Transformation Map process. Over a period of use, a number of directories and files may accumulate here. You may clear the contents of this directory to save space. You may do this at any time providing no transformation maps are being prepared at the time.

## 8.2.4 Browser

LANSA Composer implements a web-browser based interface for operational monitoring and control of the application. For convenience of the LANSA Composer designer, however, the browser interface is embedded in the Composer client software. The URLs in this section are used to display the browser interface in the client software.

You can use the *Verify URL* button to open the specified URL in your web browser in order to verify that the URL is correct and operating.

🔯 System settings	
System Settings Browser Base URL for browser interface URL for LANSA Integrator (JSM) console SMTP mail details defaults	Base URL for browser interface- Value: http://youriSeries/cgi-bin/lansaweb? Verify URL
From e-mail address From e-mail display name From e-mail display name HTTP inbound processing HTTP inbound payload directory HTTP inbound file name prefix Default Configuration	Hint- This value specifies the root url used from the client computer to access the web browser based operations console. LANSA Composer adds the application-specific portion of the URL.
Save Refresh	

### Base URL for browser interface

# Example: http://youriSeries/CGI-BIN/lansaweb?

This value specifies the base URL used from the client computer to access the web-browser based operations console for LANSA Composer. It is used to construct the URL used when the *Console* option is chosen from the *Operations* section in the *Navigator*.

### URL for LANSA Integrator (JSM) console

### Example: http://youriSeries:4561

This specifies the URL used from the client computer to display LANSA Integrator's Java Service Manager Console. It is used to display the console when the *Java Service Manager Console* option is chosen from the *Operations* section in the *Navigator*.

The URL is usually in the form http://host:port , where host is the name of the application server on which the Java Service Manager is running and port is the port number specified for the console in LANSA Integrator's manager.properties file (default 4561). Refer to the Java Service Manager Console in the *LANSA Integrator Guide* for more information.

# 8.2.5 SMTP Mail Details Defaults

The values in this section specify defaults that are pre-filled when a new SMTP mail detail configuration is created.

M aystem settings	
System Settings SMTP mail details defaults From e-mail address From e-mail display name HTTP inbound processing HTTP inbound processing HTTP inbound of directory HTTP inbound of the set o	From e-mail address Value: fromaddress@yourcompany.com Reset Value
Other settings	Hint This value is used when an SMTP mail detail configuration is created. It is used to set the initial "From email address". The value may be changed in individual SMTP email detail configurations.

#### From e-mail address

Example: Fred.Smith@yourcompany.com

This value is used when an SMTP mail detail configuration is created. It is used to set the initial *From email address*. The value may be changed in individual SMTP email detail configurations.

#### From e-mail display name

Example: Fred Smith

This value is used when an SMTP mail detail configuration is created. It is used to set the initial *From email Name*. The value may be changed in individual SMTP email detail configurations.

# 8.2.6 HTTP Inbound Processing

The values in this section specify defaults that are used by the supplied HTTP Inbound Activity, HTTP\_INBOUND.

🔯 System settings		
System Settings	^	HTTP inbound payload directory-
🖃 🌏 HTTP inbound processing		Value:
HTTP inbound payload directory		/LANSA_Composer/HTTPIN Browse
HTTP inbound file name prefix		
Default Configuration		Reset Value
SMTP server configuration		
SMS configuration		Lint
Other settings		
Archive processing sequence versions		This value is used by the supplied HTTP Inbound Activity, HTTP_INBOUND. It is used
Zip archive file name prefix		to identify the directory into which the payload of HTTP Inbound messages are
POP3 save sub-directory name prefix	×	saved. If you are going to be using inbound hit is processing, then a directory on
		your server must be entered.
💾 Save 🛛 🐯 Refresh		

### HTTP inbound payload directory

### Example: /Composer/HTTPIN

This value is used by the supplied HTTP Inbound Activity, HTTP\_INBOUND. It is used to identify the directory into which the payload of HTTP Inbound messages are saved. If you are going to be using Inbound HTTP processing, then a directory on your server must be entered.

### HTTP inbound file name prefix

Example: HI

This value is used by the supplied HTTP Inbound Activity, HTTP\_INBOUND. It is used as the prefix for the name of the saved payload of HTTP Inbound messages. This prefix plus a sequential number will be used as the file name of the saved payload. The file will be saved in the HTTP Inbound directory.

For example: The payload of an inbound HTTP message would be saved as /Composer/HTTPIN/HI00000123.dat

## 8.2.7 Transaction Document Processing

The values in this section specify values used in connection with LANSA Composer's transaction document processing framework and features.

stem Settings		"Resend" processing sequence
Transaction document processin "Resend" processing sequence	g	Select:
Default configuration		
SMS configuration		
Other settings Java version for Transformation Maps	=	Hint
Archive transformation map versions Archive processing sequence versions		This value specifies the processing sequence that processes the "Resend" button when clicked in the LANSA Composer Document Manager. The
Zin archive file name prefix		processing sequence should be modelled on the supplied 1XDOC_SEND processing sequence.

### "Resend" processing sequence

# Example: TXDOC\_SEND

This value specifies the name of the processing sequence that LANSA Composer will execute to service the Resend button, when clicked, in the LANSA Composer Document Manager application. The supplied processing sequence, TXDOC\_SEND, is the default choice. If you change this to a processing sequence of your own, then it should be modeled on TXDOC\_SEND with respect to the parameters it receives. You may click the adjacent prompt button to select from a list of available processing sequences.
# 8.2.8 Default Configuration

The values in this section identify transport configurations set as defaults for event notifications and certain supplied activities.

<ul> <li>System settings</li> <li>System Settings</li> <li>Offault Configuration</li> <li>SMTP server configuration</li> <li>SMS configuration</li> <li>SMS configuration</li> <li>Other settings</li> <li>Archive processing sequence versions</li> <li>Zip archive file name prefix</li> <li>POP3 save sub-directory name prefix</li> <li>From e-mail address for event notific</li> <li>Job description for Log Purge job</li> </ul>	SMTP server configuration Select: DEFAULT_SMTP Hint This value specifies an SMTP server configuration that provides default SMTP server details for email event notifications and for some email-related activities.
E Save Refresh	

#### **SMTP** server configuration

This value specifies an SMTP server configuration that provides default SMTP server details for email event notification and for other supplied activities which may require SMTP server details. For example, an SMS configuration can refer to the default SMTP server configuration by selecting *(Default)* from the drop down list of available SMTP server configurations.

#### **SMS configuration**

This value specifies an SMS configuration that provides default SMS configuration details for SMS event notification or other SMS related activities such as the supplied SMS\_SEND activity.

# 8.2.9 Other Settings

This section contains various settings which are used as the default for values throughout Composer.

🔯 System settings	
System Settings	From e-mail address for event notifications
Oefault Configuration     SMTP server configuration     SMS configuration	value: notification@yourcompany.com Reset Value
Computation     Computati	
POP3 save sub-directory name prefix From e-mail address for event notificati	Hint Emails sent by the Composer processing when a failure is encountered and an Event notification exists for that failure must have a From email address for email validity. The value
Save Save	entered here is used as that from email address.

#### Java version for Transformation Maps

This setting determines the target Java version used when compiling the generated Java code for Transformation Maps. You should select the Java version that is compatible both with the version assumed by the mapping tool and with the JVM and LANSA Integrator on the LANSA Composer server that will execute the Transformation Maps.

#### Archive processing sequence versions

When enabled, this setting causes LANSA Composer to archive prior Processing Sequence versions each time a modified Processing Sequence definition is saved in the *Processing Sequence Editor*. This enables the prior versions to be restored, if required.

#### Zip archive file name prefix

Example: ZP

This value is used by the supplied Activity ZIP\_LIST, which zips a list of files. If a zip archive name is not provided to this Activity, it will create an archive name consisting of this prefix plus a sequential number. For example: ZP0000456

#### POP3 save sub-directory prefix

Example: MS

This value is used by the supplied Activity MAILRECEIVE\_ALL which gets all the email messages from the mail server. The mail details and attachments

are saved into a directory specified on the configuration or at processing time. The information from each mail is saved into a sub-directory which is created. This sub-directory is named with this prefix plus a sequence number.

#### From e-mail address for event notifications

#### Example: Composer.processing@yourcompany.com.au

Emails sent by the Composer processing when a failure is encountered and an Event notification exists for that failure must have a *From* email address for email validity. The value entered here is used as that *From* email address.

#### Job description for submitted jobs

#### Example: QBATCH.QGPL

This value is used as the job description when submitting jobs such as Processing Sequence runs and Database Housekeeping.

#### Show supplied definitions for export

This setting is disabled by default. When enabled, supplied processing sequences, transformation maps and activities will be shown in the LANSA Composer export dialogue. This setting need to be enabled in order to export supplied definitions.

# **8.3 System Properties**

System properties allows you to extend the standard LANSA Composer environment with properties that are specific to your own installation, environment or organisation. Using system properties you can:

- Define your own system properties for use with LANSA Composer
- Assign or change values for your system properties or specify a LANSA function that will evaluate your system properties
- Optionally, specify that a system property is writeable, thereby permitting your Processing Sequences to assign a new value to the system property.

Having defined your own system properties, you can refer to them in your Processing Sequences using built-in variables of the form:

\*SYSTEM.PROPERTY.<property name>

where <property name> is the name of the installation-defined system property.

The *Built-ins* tab in the *Processing Sequence Editor* will show built-in variables for any system properties that you have defined here.

For information about accessing system property values in a processing sequence, refer to Using the System Property Built-in Variables .

For information about writing a system property evaluation function refer to 8.3.1 System Property Evaluation Functions.

To reach the system properties, expand *Administration and Housekeeping* in the *Navigator* and then select the required function.

System propertie	8			
System property n	System property description			
	Test data input files location	System property name	IC_DATA_IN	
1C_DAIN_001	rest data ouput nes location	System property description	Test data input files location	
		-System property value:		
		Use this value:		<b>.</b>
		/LANSA_Composer_xf3pgmli	b/xf3/TC_DATA_IN	
		Processing sequences car	n set the value (writeable)	
		Save		
	New	Delete		

To work with system properties, select the required property from the list. You can use the *New* or *Delete* buttons to add or remove system properties, or you can complete the following details for new or revised system property.

Remember to press the *Save* button to save your changes.

System property name	Specifies a name for the system property. The name must be unique amongst system properties in this installation of LANSA Composer. The name can be up to 20 characters, must begin with an alphabetic character in the range A-Z and consist only of alphabetic characters in that range, digits 0-9 and underscore (_).
System property description	Specifies a text description for the system property. It can be up to 50 characters.
System property	The entry fields in this section specify how the system property value is to be determined or derived.
value	In the drop-down list you can choose from two evaluation methods:
	<i>Use this value:</i> When you select this option, the second entry field specifies the value for the system property. It can be up to 256 characters.
	<i>Evaluate / set by call to function:</i> When you select this option, the second entry field specifies the name of a LANSA function that will be called to evaluate the system property value each time it is referenced (and to set its value when a new value is assigned to a <i>writeable</i> system property).
	For information on writing system property evaluation functions refer to 8.3.1 System Property Evaluation Functions.
Processing sequences can set the value (writeable)	Check this box to permit Processing Sequences to assign a new value to the system property.

# **8.3.1 System Property Evaluation Functions**

If you choose the option to *Evaluate / set by call to function*, then it is your responsibility to create and deploy or install the specified LANSA function that will evaluate (and set) your system property's value.

- Your function may be created using RDML or it may be fully RDMLX enabled.
- Your function must be defined with *function options(\*direct)*
- Your function must be created and compiled at a LANSA level that is compatible with the version of LANSA Composer that you are using
- Your function must be deployed to the partition module library for the LANSA system and partition in which LANSA Composer is installed on your LANSA Composer server (for example LICLICLIB)

When LANSA Composer calls your function to evaluate or set a system property value, it will communicate with it using the LANSA exchange list. Specifically, your function must define (if necessary) and use the following fields to identify the request and to return the result. In the case of fields that are indicated to be *output* variables, your function must use the EXCHANGE command to place them on the LANSA exchange list before returning.

Name	Description	Туре	Usage
DXREQUES	KREQUES Request code. LANSA Composer will set this to one of the following values:		Input
	'GET' - indicates the function should evaluate the specified system property and place its value in field DXSPVL		
	'SET' - indicates the function should set the value of the specified system property using the value provided in field DXSPVL.		
DXSPID	System property identifier. Your function may be used to evaluate or set the values of more than one system property. If you design your solution in this way, your function will need to test this value (and/or the DXSPII field value) to determine which system property it is being requested to evaluate or	A(20)	Input

set.

DXSPII	System property internal identifier. This is a unique internal identifier that LANSA Composer assigns to each system property definition.	A(32)	Input
DXSPVL	System property value. For a GET request, your function must set the value of this field to contain the evaluated value of the specified system property.	A(256)	Input (SET) Output (GET)
	For a SET request, your function should set the value of the specified system property using the value provided in this field.		
DXRESULT	Result code. If your function completes normally, its should set the value of this field to 'OK'. If the result code contains any other value, LANSA Composer will assume that the evaluation request failed and will not use the returned system property value.	A(2)	Output

The following is an example RDMLX function to evaluate (and set) a system property named MY\_PROPERTY.

function options(\*direct)

\* \_\_\_\_\_

\* Program mainline

```
* _____
```

case of\_field(#dxspid)

when value\_is(= 'MY\_PROPERTY')

case of\_field(#dxreques)

when value\_is(= 'GET')

#dxspvl := 'MY\_PROPERTY\_VALUE'
#dxresult := 'OK'
exchange fields(#dxspvl #dxresult)

when value\_is(= 'SET')

\* MY\_PROPERTY is NOT intended to be writeable #dxresult := 'ER' exchange fields(#dxresult) abort msgtxt('System property MY\_PROPERTY not writeable')

endcase

otherwise

abort msgtxt('System property name not recognised by this function')

endcase

return

# 8.4 Code Maintenance

To reach Code Maintenance, expand *Administration and Housekeeping* in the *Navigator* and then select the required function.

Code Maintenance allows you to view or modify various code values that are used in LANSA Composer to categorize and organize definitions. The codes are in these Groups:

	Used for		
Activity groups	To categorize Activities.		
	The groups to which an Activity belongs is maintained in the Activity maintenance. An Activity may belong to several groups. The grouping is used within the Processing Sequence editor to make it easier to find and select Activities.		
Transformation map types	To provide a grouping and plain language description for Map Types.		
	The groupings are used in the maintenance of Trading Partners Linked Maps. Within Processing Sequences you can select specific types of Linked Maps for the current Trading Partner.		
TradingTo categorize Trading Partners.			
partner groups	The groups to which a Trading Partner belongs is maintained in the Trading Partner maintenance. A Trading Partner may belong to several groups.		
Trading partner	To provide a grouping and plain language description for Directory Types.		
directory types	The groupings are used in the maintenance of Trading Partners Linked Directories. Within Processing Sequences you can select specific types of Linked Directories for the Trading Partner currently being processed.		
Trading partner installation-	To define properties for Trading Partners specific to your organization and the transactions you conduct with your Trading Partners.		

defined properties	Once Trading Partner properties are defined, property values can be entered for each Trading Partner.	
Transaction document	To add user-defined status codes for use with LANSA Composer Document Manager.	
status codes	Once a status code have been defined, the status code will then be recognized in LANSA Composer Document Manager.	

Code Maintenance allows you to add, change and delete code values in these groups.

Code group				
Activity groups				
Transformation map	Transformation map types			
Trading partner gro	oups			
Trading partner dire	ectory types			
Trading partner ins	tallation-defined properties			
ode mainter	nance : Maintain (-Activity	groups)		
Code Value	Code Description	Code Value	FTP	
DESIGN	Design, test and debug			
EMAIL	Email	Code Description	FTP transport activities	
FILE_MANAGE	File management			
FTP	FTP transport activities	Code Description (Short)	FTP transport	
HTTP	HTTP transport activities	and the second se		
MESSAGING	Messaging transport ac	Save		
PROCESS	Processing			
TRANSFORM	Transformations	Delete		
TRANSPORT	Tutariala	Delete		
VADIARIES	lutoriais Variable meninulation			
VARIABLES	Variable manipulation			
21P Zip activities				
	New			

To work with Code Values, select the required Code Group and then select the Code Value from the code value list. You can use the *New* or *Delete* buttons to add or remove code values, or you can complete the following details for new or revised code values. Remember to press the *Save* button to save your changes.

CodeSpecifies a text description for the code value. It can be up toDescription50 characters.

The following details apply only to the Trading partner installation-defined

properties group:

Code Value Type	Defines the code value type as Alpha or Numeric.
Code Value Length	Defines the length of the code value.
Code Value Decimals	Defines the number of decimal digits for a numeric type code value.
Code Sequence	Defines the display sequence for properties in Trading Partner Properties tab.

The following details apply only to the Transaction document status codes:

Resulting Specifies if the particular status signifies an error, that processing has completed, that the document has been abandoned or that processing is still in progress.

# 8.5 Event Maintenance

LANSA Composer provides support for event definitions that can be "triggered" during execution of a Processing Sequence and that can provide notifications of the event, for example, by email or SMS message to a system operator. These events can be set up and maintained in *Event Maintenance* (under *Administration and Housekeeping* in the LANSA Composer *Navigator*).

There are three ways that an event definition can be "triggered" during the execution of a Processing Sequence:

- 1. Your Processing Sequence can explicitly cause an event to be triggered by including the supplied NOTIFYEVENT activity. You can use this activity to trigger any defined event, including those that are supplied with LANSA Composer and additional events that you may define yourself.
- 2. You can associate a specific event with a Processing Sequence such that the event is triggered automatically if the Processing Sequence ends in error. To do this, enter the required event definition name in the *Event identifier* field at the Processing Sequence *Details* command tab
- 3. Activities can be written such that they trigger an event in specific circumstances. A number of the supplied activities behave in this way. For example, the supplied FTP\_INBOUND activity can trigger the FTPINFAILED event if an error occurs during the FTP processing. Refer to 8.5.1 Supplied Events for a list of the event definitions that can be triggered by activities supplied with LANSA Composer.

If you are creating your own custom activities, you can employ this same technique to trigger an event when appropriate. For more information refer to Signal an event .

Note: Both your own event definitions and the supplied event definitions will be triggered only if their status is set to *Active*. Supplied events have a status of *Inactive* as installed. You must enable them by setting their status to *Active* if you wish them to be triggered. Of course, if you do this,, you should modify the supplied events to notify you of the associated event by whatever means is appropriate for your organization.

When an event is triggered in any of the above ways, the Processing Sequence

controller will notify the appropriate event to provide the notifications that are defined for the event.

Refer to 8.5.2 Event Details for information on the basic details for an event definition and on adding, removing and organizing the notification(s) for the event.

An event can use one or more of the following notification methods to provide the notification of the event:

- 8.5.3 Send an Email
- 8.5.4 Send an SMS
- 8.5.5 Send a Message to an IBM i Message Queue (IBM i servers only)
- 8.5.6 Run a Specified Processing Sequence
- 8.5.7 Execute a Specified Function

## **8.5.1 Supplied Events**

The following are the events that are supplied with LANSA Composer to handle failures in supplied Activities:

Event	Description	Invoked by Activity
FTPCMDLSTFAILED	Notification of FTP failure using Command list.	FTP_COMMANDLIST
FTPINFAILED	Notification of inbound FTP failure	FTP_INBOUND
FTPOUTFAILED	Notification of outbound FTP failure	FTP_OUTBOUND
HTTPOUTFAILED	Notification of outbound HTTP failure	HTTP_GET HTTP_POST
MSGSENDFAILED	Notification of failure sending a message through a message brokering system such as IBM MQ Series	MSG_SEND
MSGRCVFAILED	Notification of failure receiving a message through a message brokering system such as IBM MQ Series	MSG_RECEIVE
POP3FAILED	Notification of failure reading emails with POP3	MAIL_RECEIVE MAIL_RECEIVEALL
SMTPFAILED	Notification sending email using SMTP	MAIL_SEND
SMSFAILED	Notification of failure while sending SMS	SMS_SEND
TRANSFORMFAILED	Notification of error executing a transformation map.	<b>TRANSFORM</b> activity or any Transformation Map directive

See the Notes tab of the supplied events for a description. You can enable or disable these supplied events by swapping their *Status* between *Active* and *Inactive*. You can modify the supplied events to notify you of the associated event in whatever way is appropriate for your organization.

# 8.5.2 Event Details



An event may be set up with:

**ID** A unique name for the event

**Description** Text description for the event.

**Status** Active or Inactive. If the event status is not Active, then the event will not be triggered.

#### Define and organize notifications

For each event definition, you may add one or more notifications. Use the *Add*, *Remove*, *Move up* or *Move down* buttons to define and organize the notifications.

You can add a notification but disable it by unchecking the checkbox next to the notification in the list.

You can add more than one notification of the same type, if you wish.

When you click the *Add* button, LANSA Composer shows the *Choose Event Notification Method* window to allow you to choose what type of notification to add:

Choose Event	Notification Method
	Choose the event notification method:
	OK

Refer to the following headings for more information about each notification type:

- 8.5.3 Send an Email
- 8.5.4 Send an SMS
- 8.5.5 Send a Message to an IBM i Message Queue (IBM i servers only)
- 8.5.6 Run a Specified Processing Sequence
- 8.5.7 Execute a Specified Function

## 8.5.3 Send an Email

The following details may be entered for the *Send an email* event notification method:

- **TO** This value must be provided. It should contain a valid email address, or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.
- **CC** This optional value specifies the CC (copy to) address. It can contain a single email address, or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.
- **BCC** This optional value specifies the BCC (blind copy) address. It can contain a single email address, or a list of email addresses separated by commas. Click the adjacent prompt button to enter or maintain the list of email addresses.
- **Subject** A subject must be entered. This will be the subject for the email sent.

# IncludeYes or No. Yes will result in the sent email containing a LogLognumber reference in the email text. That Log reference can then bereferenceused by the email recipient to look up the details using thein textProcessing Log Inquiry.

**Text** You can provide text for the body of the email. A short description for the reason for the email is suggested.

Note: Email notifications use the LANSA Integrator SMTP email facilities. To use the Event emails you must have LANSA Integrator running and a valid *Default SMTP configuration* set in *System Settings* that refers to your SMTP mail server.

# 8.5.4 Send an SMS

The following details may be entered for the *Send an SMS* event notification method:

TO<br/>SMS<br/>numberThis parameter may contain the SMS number to which the message<br/>is to be sent<br/>numberSMS<br/>MessageThis parameter may contain the SMS message text to be sent.SMS<br/>MessageYes or No. Yes will result in the sent SMS message containing a<br/>Log number reference in the message text. That Log reference can<br/>then be used by the message recipient to look up the details using<br/>the Processing Log Inquiry.

Note SMS notifications use the LANSA Integrator SMS and SMTP email facilities. To use the event SMS you must have LANSA Integrator running, and a valid *Default SMTP configuration* and *Default SMS configuration* set in *System Settings* that refers to your SMTP mail server and to your SMS provider. If your SMS provider expects email in a format different to that generated by the standard LANSA Composer SMS support, then you may need to implement that with a custom solution.

#### 8.5.5 Send a Message to an IBM i Message Queue

NOTE: This notification method is supported only when LANSA Composer is running on an IBM i server.

The following details may be entered for the *Send a message to an IBM i message queue* event notification method:

Message<br/>textThis parameter may contain the message text to be sent to the<br/>specified message queue.Message<br/>queue<br/>name<br/>andSpecifies the IBM i message queue to which the message is to be<br/>sent. You may specify the special values \*SYSOPR or \*HSTLOG<br/>in the Message queue name to send the message to the system<br/>operator message queue or to the history log respectively.

## 8.5.6 Run a Specified Processing Sequence

The following details may be entered for the *Run a specified processing sequence* event notification method:

LANSA	Specifies the name of a LANSA system configuration that
system	will be used to run the specified Processing Sequence
configuration	(through the LANSA Composer Request Server). Usually the
	LANSA system configuration should refer to the same
	LANSA Composer system that contains the Processing
	Sequence that triggered the event.
	You can use the prompt button to choose the LANSA system configuration from a list of available LANSA system configurations.
Processing sequence	Specifies the name of the Processing Sequence that the event should run.

You can use the prompt button to choose the Processing Sequence from a list of available Processing Sequences.

When the Processing Sequence runs, LANSA Composer will make the following parameters available to it, containing further information about the event that was triggered and the processing sequence and error that triggered it:

DXEVID Event name

DXP1ID Processing sequence name

DXL1NO Run number

DXL3SD Short error description (if any)

DXL3TX Long error text (if any)

An example event notification handler Processing Sequence is supplied with LANSA Composer. If you have installed the supplied examples, refer to the Processing Sequence EXAMPLE\_EVENT01.

Note The specified Processing Sequence will be run (synchronously) through the LANSA Composer Request Server to ensure that it runs in a separate process to the Processing Sequence that triggered the

event. If you haven't already done so, you may need to configure the LANSA Composer Request Server for your system. Refer to Appendix F. The LANSA Composer Request Server for detailed information about considerations for requests executed through the LANSA Composer request server.

## **8.5.7 Execute a Specified Function**

The following details may be entered for the *Execute a specified function* event notification method:

FunctionSpecifies the name of a user-defined function that is to be<br/>executed.

NOTE: To make use of this notification method, you will need access to LANSA development tools and skills in order to code, compile and install the necessary user-defined function.

Note the following about the function to be called:

- The function must be installed in the LANSA Composer server system in the partition containing LANSA Composer (normally LIC).
- The function must be coded using RDMLX.
- The function must have been compiled in a LANSA development environment that is at a level that is compatible with the LANSA Composer system.
- The function must be defined with *options(\*DIRECT)*.
- The function must not directly invoke any LANSA Composer entry point. For example, it must not attempt to (directly) execute another Processing Sequence.

Before the function executes, LANSA Composer will populate the exchange list with the following fields that the function may reference, if desired:

DXEVID	A(10)	Event name
DXEVII	A(32)	Event internal identifier
DXP1ID	A(10)	Processing sequence name
DXP1II	A(32)	Processing sequence internal identifier
DXPVRN	A(15)	Run number as a character string
DXL1NO	P(15, 0)	Run number
DXL3MN	A(7)	Last error number, if any
DXL3SD	A(80)	Last error short text description, if any

DXL3TX S(512) Last error long text description, if any

An example of a user-defined event handler function is supplied in the LANSA Composer development package in process DXSXTC, function DXEVEN1. Refer to Before you begin your custom activity .for information on how to locate and import this package into your LANSA development environment.

# 8.6 Database Housekeeping

Database Housekeeping is available under *Administration and Housekeeping* to maintain and reorganize the LANSA Composer database.

The database housekeeping screen displays when the housekeeping tasks were last run, as well as permitting you to enter the values to be used for future database housekeeping tasks.

After entering the required values, there are two ways to proceed:

1. Save the values and submit the database housekeeping task from the command line

This technique allows you to integrate the database housekeeping task into your other operational procedures, for example, by scheduling it with a job scheduler. Remember that the database housekeeping task will need to run at a time when exclusive use of the LANSA Composer system will be available to it.

You can click the *Save* button to save the values that will be used for any future database housekeeping tasks.

Refer to Appendix D Commands to invoke LANSA Composer for information about running the database housekeeping task on your LANSA Composer IBM i or Windows server using an appropriate command line interface.

2. Submit the task directly from the LANSA Composer client

Click the Submit button to submit the task directly from the LANSA Composer client using the values presently displayed.

NOTE: It is recommended that the submitted Database Housekeeping job should have exclusive access to the LANSA Composer system. For this reason, the LANSA Composer client application will close after submitting the Database Housekeeping task.

The Database Housekeeping job will log the main actions that it has performed into the LANSA Composer log files. You can click the *History* tab to display or print the logs for completed Database Housekeeping tasks.

Database housekeeping : Submit	
Submit History	
Do you wish to purge the logs?	
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you want to keep entries with error status?	No
Do you wish to retain entries with warning status?	No
Do you wish to purge the version history?	V
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you wish to purge the transaction document register?	
Last purged date and time	2013-08-06 8:00:48 AM
Number of days for which you wish to retain records	180
Do you wish to purge the audit records?	
Last purged date and time	2013-08-06 8:00:49 AM
Number of days for which you wish to retain records	180
Do you wish to clear the ISM trace and temporary files?	
Last number date and time	2013-08-06 8-02-48 AM
Lost pargeo dote and ane	2010 00 00 000102 10 411
Do you wish to reorganize the database tables?	
Last reorganised date and time	2013-08-06 8:02:48 AM
Do you wish to reconcile the database ?	
Last reconciled date and time	2013-08-06 7:56:58 AM

Do you<br/>wish to<br/>purge the<br/>logs?Check to confirm. The related fields (described below) that<br/>control how the log is purged apply only when this box is<br/>checked.Last<br/>purged<br/>date andDisplays, for your information, when the log was last purged

time	
Number of days for which you wish to retain records	You may enter the number of days for which the records should be retained. All entries older than this number will be purged.
Do you want to keep entries with error status?	Select Yes to retain log entries for Processing Sequence runs or system tasks that resulted in an error. Select No to purge log entries irrespective of their error completion status if they meet the other purge criteria.
Do you want to keep entries with warning status?	Select Yes to retain log entries for Processing Sequence runs or system tasks that resulted in a warning. Select No to purge log entries irrespective of their warning completion status if they meet the other purge criteria.
Do you wish to purge the version history?	Check to confirm. The related fields (described below) that control how archived processing sequence and transformation map versions are purged apply only when this box is checked.
Last purged date and time	Displays, for your information, when processing sequence and transformation map versions were last purged.
Number of days for which you wish to retain records	You may enter the number of days for which archived processing sequences and transformation maps should be retained. All archived versions older than this number will be purged.

Do you wish to purge the transaction document register?	Check to confirm. The related fields (described below) that control how transaction document register entries are purged apply only when this box is checked.
Last purged date and time	Displays, for your information, when transaction document register entries were last purged.
Number of days for which you wish to retain records	You may enter the number of days for which transaction document register entries should be retained. All entries older than this number will be purged.
Do you wish to purge the audit records?	Check to confirm. The related fields (described below) that control how audit trail records are purged apply only when this box is checked.
Last purged date and time	Displays, for your information, when audit records were last purged.
Number of days for which you wish to retain records	You may enter the number of days for which audit records should be retained. All entries older than this number will be purged.
Do you wish to clear the	Check to confirm. This option will clear trace and temporary files that have been created by the LANSA Integrator JSM (Java Service Manager).

JSM trace and temporary files? Last purged date and	Displays, for your information, the last date when the JSM trace and temporary files were cleared.
time	
Do you wish to reorganize the database tables?	Check to confirm. This option applies to LANSA Composer on IBM i servers only. It will perform a physical reorganization (RGZPFM) of the LANSA Composer database tables.
Last reorganized date and time	Displays, for your information, the last date when the database was reorganized.
Do you wish to reconcile	Check to confirm. This option will remove redundant entries in the LANSA Composer database.
the database?	NOTE: This step can take a significant amount of time to run and should not be necessary in the course of normal operations. It may become desirable if abnormal terminations of LANSA Composer jobs have left orphaned records in your database, but even then such redundant records should not adversely affect LANSA Composer operation. This option is unselected by default.
Last reconciled date and time	Displays, for your information, the last date when the database was reconciled.

# 8.7 User Access Configuration

The 'Configure User Access' display allows access restrictions to be set for a specific user or group. The LANSA Composer and LANSA Composer Document Manager applications each provide their own interface for this task.

# **About User Access Configuration**

User Access Configuration can be used to customise the functionality available for each user. Using this feature, access rights can be modified such that users will only see items relevant to them.

# What it is, What it is not, Warnings and Caveats

User Access Configuration is meant to be a convenient way of simplifying a user's view of the LANSA Composer or LANSA Composer Document Manager application. It is not meant to be a used as a security mechanism to prevent users from viewing or modifying sensitive information.

Refer to the following for more information:

8.7.1 Using User Access Configuration

## 8.7.1 Using User Access Configuration

To access this feature, select *Tools* from the menu bar and then *Configure User Access* in either LANSA Composer or LANSA Composer Document Manager.If you do not see this menu option, that is usually because it has been disabled by the administrator. If there are issues, please contact the system administrator or other relevant parties.

S Configure user access	
1. Type a username or choose one from the list	
LICPGMLIB	✓ User can modify access rights
	User access restriction applies for this user Arefresh
1 Oblic	▼ User has access to all functions
	V User has access to all commands
<ol> <li>Choose the LANSA Composer function to which you wish to grant or deny access.</li> </ol>	3. Grant or deny access to the Activities function
<ul> <li>□ LANSA Composer</li> <li>□ → Operations</li> <li>□ Console</li> <li>□ Java Service Manager console</li> <li>☆ Run processing sequence shortcuts</li> <li>□ O Definitions</li> </ul>	(User has access to all functions including this function.)
Activities     Same Trading partners     Transformation maps     Processing sequences	Delete     Details (Default)     Groups     New
<ul> <li>Configurations</li> <li>FTP configurations</li> <li>HTTP configurations</li> <li>SMTP server configurations</li> <li>SMTP mail details</li> <li>POP3 mail configurations</li> <li>SMS configurations</li> <li>Messaging configurations</li> </ul>	<ul> <li>Notes</li> <li>Parameters</li> <li>Run history</li> <li>Where used</li> <li>Print</li> <li>Print list</li> <li>Run</li> </ul>

The Configure User Access window is shown and allows you to set access restrictions in four simple steps. These steps are outlined below.

## 1. Select a User

1. Type a username or choose one from the list
LICPGMLIB
LICPGMLIB *PUBLIC

First, select a user by typing in the username in the textbox, followed by the Enter key or the refresh button. Alternatively, one can be selected from the

given list. This user can be refer to a user profile, a group profile (on IBM i server) or '\*PUBLIC'. A group profile only applies to IBM i server and will change access rights for everyone in that group. '\*PUBLIC' applies for both Windows and IBM i server and will modify settings for all users. In the event that a setting has been set at multiple levels, settings for group will override '\*PUBLIC' setting while setting for user profile will override both.



Immediately to the right of the user selection area are four checkboxes. These checkboxes controls global access rights for the user. Their function are as follows:

- The "User can modify access rights" check box will control whether said user have access to this interface.
- The "User access restrictions apply for this user" determines whether the restrictions set below will take effect. With exception to the above checkbox, this box needs to be checked for any restriction set here to take effect.
- The "User has access to all functions" check box is a default setting for function visibility. Ideally, this should be unchecked when a majority of functions is to be disabled and checked otherwise. Access to individual functions can then be 'granted' or 'denied' as necessary.
- The "User has access to all commands" check box is a default setting for all commands. This should be unchecked if users is to be denied access to a majority of commands and checked otherwise. Access to individual commands within each function can then be individually granted or denied as necessary. Do note that default commands cannot be disabled and LANSA Composer may re-enable them individually if necessary.

•

## 2. Select a Function



Next, select a relevant function from the tree on the left hand side that shows all the functions available in the relevant application (LANSA Composer or LANSA Composer Document Manager). This will populate a drop down on the right and possibly also the list view.

## 3. Grant or deny access to Functions



The drop down will have three entries by default and will control whether or not this function will be visible when the said user logs in. The first, default entry means that access rights to this function (and all functions) is controlled universally by the "User has access to all functions" checkbox. This means access can be 'granted' or 'denied' for all functions via said check box except for those that have been explicitly set. The second and third entry allows access to be 'granted' or 'denied' specifically for the function.



## 4. Grant or deny access to commands

Most functions will have commands associated with them that identify the range of operations that can be performed for that function or for an instance of that type of definition (for example, for a selected Trading Partner). Commands will reflect things that users can do usually through the command tabs shown for the function or by clicking one of the toolbar buttons or by right-clicking an instance. Commands can include such things such as New and , Copy, Details, Parameters and Notes, depending on which of the LANSA Composer functions is selected..

If there are commands for the function, the list below the drop box will have several entries, each with a check box. Each item in the list represents a command for the selected function. By toggling the checkboxes, access rights can be given or revoked accordingly. By default, checkboxes will be grayed, indicating that access to commands is controlled (granted or revoked) universally, using the "User has access to all commands" checkbox.

Do note that Default commands, as indicated by '(Default)' tag cannot be disabled.

## Save and Activate Changes



To save all changes so far, click the save button. Note that for these changes to take effect, the *User access restriction applies for this user* option needs to be checked (selected) before saving. If not selected, users would be exempt from an authority check, giving them access to all functions and commands.

Alternatively, the *Refresh* button can be used to clear all unsaved changes. Clicking *Refresh* will reload currently saved settings for the user. To undo all the changes and stop modifying access rights, use the *Close* button. As a safety net, a warning message will pop up asking whether changes are to be saved. To proceed, click **No**.

You can also delete both saved and unsaved changes using the *Delete* button. In addition to clearing changes in the user interface, the *Delete* button will also clear all previously saved changes. Data will then be reloaded as per normal. As data for the user would have been cleared, loaded data will be based on group settings, '\*PUBLIC' settings and default settings, whichever ones may apply to the user.

# 9. Develop Custom Activities for LANSA Composer

You can define your own custom Activities for use with LANSA Composer. Once your Activity is fully-described to Composer, you can orchestrate it along with other Activities and Transformation Maps in a Processing Sequence.

You will require Visual LANSA software and have the ability to write Visual LANSA RDMLX programs in order to create your own Activities.

Developing a custom Activity should involve at least the following steps:

9.1 Plan Your Custom Activity

9.2 Define the Activity to LANSA Composer

9.3 Create the Activity Processor

9.4 Test your Activity

9.5 Deploy your Custom Activity

For general considerations about integrating your custom LANSA code with your LANSA Composer solution refer to:

Appendix E. Using LANSA Composer with LANSA Applications
# 9.1 Plan Your Custom Activity

You should spend some time planning your custom Activity before beginning development. Some of the things you need to think about include:

• **Inputs and outputs:** what variable information will this Activity require as input and create as output?

Your Activity Processor can receive variables and variable lists from the variable pool for the Processing Sequence in which it is running. Equally, it can place variables and variable lists on the variable pool. Variables and variable lists are the means by which your Activity can "communicate" with other Activities and transformations in a Processing Sequence. By receiving and returning variable information to and from the variable pool, you can make your Activity more flexible and able to be used for a wider variety of cases.

• **Performance:** how much (or little) work should your custom Activity perform in a single invocation?

Think about your Activity as a logical "unit of work" that you may wish to combine with other custom Activities, supplied transport and other Activities and/or transformations.

From the performance point of view, the more work your custom Activity does the better. The processing your custom Activity performs is typically much more efficient than time spent in the Processing Sequence engine switching between Activities. If your custom processing will be used in a high-volume, high-throughput application, you should plan to implement it in fewer Activities.

• **Exception handling:** what exceptions might your Activity encounter and how should you prepare for them and handle them? Can or should your Activity implement support for being restarted?

You need to understand the capabilities and limitations of the exception handling mechanism provided by the directive CATCH. CATCH is included in a LANSA Composer Processing Sequence and you need to plan how your Activity will interact with it. You also need to plan how you can best use the logging features in LANSA Composer in order to provide adequate diagnostic information in the event of an exception.

# 9.2 Define the Activity to LANSA Composer

You must define your custom Activity within LANSA Composer before it will be available for use in Processing Sequences. When you define your Activity you will specify:

- The name of the Activity Processor that implements it.
- The parameters that your Activity will receive or return.
- Other attributes that affect its execution or the way in which it can be used.

When complete, this definition allows you to orchestrate your Activity along with supplied Activities and transformations in Processing Sequences.

Refer to Work With Activities for information on defining Activities to LANSA Composer.

#### 9.3 Create the Activity Processor

The Activity Processor is the executable code that implements the function of the custom Activity. LANSA Composer invokes the processor as required in order to execute the Activity.

For all Activities that are supplied with LANSA Composer, the Activity Processors are also provided and ready to use. For any custom Activities you define to LANSA Composer, you must provide the Activity Processor.

The Activity Processor is implemented as a LANSA RDMLX component (reuseable part). You will need LANSA development skills in order to develop an Activity Processor. Refer to 9.6 Develop a Custom Activity Processor for more information.

NOTE: LANSA Composer can generate skeletal RDMLX code for your activity processor to get you started. Refer to 9.6.2 Generate Skeletal RDMLX Code for more information.

# 9.4 Test your Activity

Typically, you will test your custom Activity by creating and running Processing Sequences containing it. (If it makes sense for your activity, you can also run the Activity directly, without creating a Processing Sequence.)

Clearly you need to test that the custom processing performed by your Activity is performed correctly to your specifications in all cases. You also need to verify that your custom Activity is a good "Processing Sequence citizen". Some of the questions to ask include:

- Does it correctly set the Activity return code for normal and abnormal completion cases?
- Does it interact correctly with the variable pool? Does it clear any output variables and lists early in its processing to ensure that data from earlier invocations is not left if the current invocation ends prematurely?
- Does it provide clear and sufficient log output for normal operational monitoring and for diagnostic purposes in the event of an exception?
- If you have chosen to permit your activity to be *restartable*, does the implementation properly support the restart capability for all possible exceptions?
- Does your Activity exhibit acceptable performance characteristics for the anticipated usage? If you anticipate that the Activity will be invoked frequently in a single Processing Sequence, have you or could you take proper advantage of the *Keep Active* capability?

When testing your custom Activity, you may find some of the supplied Activities helpful. For example, you can use the LOG\_VARIABLE activity to record the values of variables in the variable pool before and after your custom Activity runs.

# **9.5 Deploy your Custom Activity**

Once your custom Activity is complete and tested, you will need to deploy it to the production environment(s) in which it will be run.

You can use the export and import features of LANSA Composer to deploy the Activity definition and any Processing Sequences that use it.

It is your responsibility to deploy the Activity Processor to the necessary target execution environments. You may be able to use LANSA's export and import support or the Visual LANSA Deployment Tool for this task, depending on your server operating environment.

Refer to Deploying Solutions For LANSA Composer for more information.

#### 9.6 Develop a Custom Activity Processor

Your Activity Processor must be implemented as a LANSA RDMLX component (re-useable part) that will reside and execute in the same LANSA system and partition as LANSA Composer.

For general considerations about integrating your custom LANSA code with your LANSA Composer solution refer to:

Appendix E. Using LANSA Composer with LANSA Applications

You must use DXACTBAS1 as the ancestor of your component. Component DXACTSKEL is provided as a skeleton Activity Processor that you can copy and adapt to your requirements.

DXACTBAS1 and DXACTSKEL are provided with the LANSA Composer software.

#### Also see

9.6.1 Before you Begin your Custom Activity

9.6.2 Generate Skeletal RDMLX Code

9.6.3 Names

9.6.4 The Ancestor Class - DXACTBAS1

9.6.5 Load and Unload

9.6.6 Initialize, Terminate and Execute

9.6.7 Access the Variable Pool

9.6.8 Understand Activity Parameters

9.6.9 Implement an Activity Processor for an Iterator Activity

9.6.10 Supporting Restartable Activities

9.6.11 Use the Java Service Manager

9.6.12 Signal an Event

9.6.13 Set the Activity Return Code

9.6.14 Use Logging Services

# 9.6.1 Before you Begin your Custom Activity

Before you begin development, you must prepare your development environment. You will need a correctly licensed and configured Visual LANSA development environment at the level defined in Install LANSA Composer.

- Development of your custom Activity Processor must be performed in a partition that is enabled for full RDMLX.
- You must import the LANSA Composer development package into the partition in which you will develop your Activity Processor. This provides:
  - base classes that you will need to compile your Activity Processor and
  - sample source code you can use to get started.

The LANSA Composer development package can be found in the folder DEV\Import on your LANSA Composer software DVD.

To get started, start the Visual LANSA development environment in the designated partition and copy the DXACTSKEL component to a new re-useable part for your Activity Processor.

# 9.6.2 Generate Skeletal RDMLX Code

LANSA Composer can generate skeletal RDMLX code for your activity processor to get you started. To do this display the activity details in the LANSA Composer client, click the *Generate* button and follow the prompts.

You will get the best results from the generated RDMLX code if you perform the generation when the activity definition is as close to final as possible.

#### **9.6.3 Names**

All Activity Processors supplied with LANSA Composer have names that begin with "DXACT". Other components of the LANSA Composer software have names that begin with "DX".

Do not use the DX prefix in your own Activity Processor names. If you do, future versions of LANSA Composer may overwrite them.

# 9.6.4 The Ancestor Class - DXACTBAS1

All Activity Processors must use DXACTBAS1 as their ancestor.

function options(\*direct)
-begin\_com role(\*extends #DXACTBAS1)

The ancestor class provides a set of properties, events and methods that you can reference in your Activity Processor that provide base services including:

- Accessing the variable pool
- Using the Java Service Manager
- Setting the Activity return code
- Logging services

Feature help is available for the base DXACTBAS1 class and for the logging helper, DXPSRLOG1. Feature help provides brief descriptions of a LANSA component's properties, methods and events. If the feature help for an item describes it as for "internal use only", then you should not use that item in your own Activity Processors.

In the Visual LANSA development environment, to get started exploring the features of the base class, put your cursor on #DXACTBAS1 in the BEGIN\_COM statement (near the top of your Activity Processor source code) and press F2.

# 9.6.5 Load and Unload

You can redefine two methods of the DXACTBAS1 ancestor class to perform one-off tasks when your Activity Processor is loaded and unloaded by the Processing Sequence controller.. These methods are:

- ActivityLoad
- ActivityUnload

These methods may become particularly significant when your Activity is defined with the *Keep Active* attribute. In this case, for a Processing Sequence that executes the activity more than once, the sequence of method calls in the Activity Processor is like this:

- ActivityLoad (the first time the Activity is executed)
  - ActivityInit ... ActivityRun ... ActivityTerm (for each time the Activity is executed)
  - ActivityInit ... ActivityRun ... ActivityTerm (for each time the Activity is executed)
  - Etc ...
- ActivityUnload (when the Processing Sequence ends)

Note: The *Keep Active* attribute does not guarantee that your Activity Processor will be loaded and unloaded only once. Under some circumstances, including specifically when the Activity return code indicates an exception, the Processing Sequence controller will unload the Activity Processor (and reload it again, if necessary, for subsequent instances of the Activity).

Defining your Activity with the *Keep Active* attribute (and implementing your Activity Processor to support it) can offer the potential of significant performance benefits for Activities:

- That are likely to be used more than once in a Processing Sequence run, and
- That have a significant load and/or unload cost (in performance terms) that need not be repeated for every execution of the Activity

For example, an Activity that exclusively uses a single LANSA Integrator (JSM) service could be written to load and initialize the service and then use the

same connection handle for all subsequent executions in the same Processing Sequence. In this event, it might redefine the ActivityUnload method to unload the JSM service and close the JSM connection.

Note: The Activity Processor must be written to fully support this mode of execution. Simply setting the *Keep Active* attribute for an existing Activity without reviewing and revising the Activity Processor may introduce unanticipated bugs. For example, the Activity Processor might have been written with the assumption that all its variables were initialized for each execution of the Activity. When you set the *Keep Active* attribute, this assumption will no longer be valid.

#### 9.6.6 Initialize, Terminate and Execute

You can redefine two methods of the DXACTBAS1 ancestor class to perform initialization and termination processing for your Activity Processor for each and every instance that your Activity is executed in a Processing Sequence. These methods are:

- ActivityInit
- ActivityTerm.

These methods are a good place to put processing such as startup and completion logging and initializing, accessing or updating variables and lists in the variable pool. You can see that the supplied Activity Processor in DXACTSKEL has suggested implementations for these methods that you can adapt to your own requirements.

It is important that you understand the distinction between methods provided for load and unload and those provided for initialize and terminate. Be sure that you place your logic in the appropriate method, having regard to your choice for the *KeepActive* attribute in the Activity definition. Refer to 9.6.5 Load and Unload for more information.

A third method of the DXACTBAS1 ancestor class, ActivityRun, is intended to be redefined in all cases to perform the main body of custom Activity

#### processing.



# 9.6.7 Access the Variable Pool

When a Processing Sequence runs, it manages a variable pool which is a pool of loosely typed named variables and variable lists that are shared across the Processing Sequence, Activities and Transformation Maps.

Your Activity Processor can access the variable pool using methods provided by the ancestor class.

```
* <you might want to retrieve values for variables
* defined as input parameters for this activity>
#std_text := #com_owner.GetVariable( '<INPUT_VARIABLE1>' )
```

Call the following methods to:

- **Clear** lists and variables, you can call these methods:
  - ClearList
  - ClearVariable
- **Retrieve** the values of variables from the variable pool:
  - GetTradingPartner
  - GetVariable
  - GetVariableRequired
- **Change** the values of variables in the variable pool:
  - PutVariable

# Variable Lists

References to variables in the variable pool can be indexed. In this way, variable lists may be created. In a Processing Sequence a list can be processed iteratively using a LOOP Processing Directive.

Any variable can be used as a list simply by indexing references to it. In your Activity Processor, you can index references to variables using the iIndex parameter of the ClearVariable, GetVariable or PutVariable methods.

Indexed references to variables should use contiguous indices starting from 1. The LOOP Processing Directive will only process instances that meet this condition.

When your Activity Processor creates a list with the PutVariable method, it will typically do so inside a loop in which you increment the value of the iIndex parameter for each call.

When your Activity Processor needs to read an existing list, it does so with the

GetVariable method inside a loop. The GetVariable method defines an output parameter, oResult, which is set to True if the variable with the specified index is found, or False if not. Thus the list processing loop can terminate when the oResult parameter is False.

# 9.6.8 Understand Activity Parameters

A part of the activity definition is the definition of the parameters that the activity expects or returns.

When writing your activity processor, it will help to understand that this part of the activity definition is more of an advertisement than a contract. It is there so that somebody using your activity in a processing sequence may set and receive values from your activity.

However, LANSA Composer does not enforce the activity parameters, nor are they automatically "passed" to the activity processor. It is the responsibility of the activity processor to retrieve variable values from the variable pool and place or update variable values in the variable pool as necessary. It is good practice to do this using the names specified in the activity parameters definition, otherwise someone using your activity in a processing sequence may not be able to set or retrieve the correct values.

#### **9.6.9 Implement an Activity Processor for an Iterator Activity**

Iterator Activities are a special form of Activity that can be used to perform iterative processing in a Processing Sequence. Refer to Iterator Activities for more information.

If you define your own Iterator Activities. you must implement the Activity Processor in a particular way such that it supports the iterative behavior. The Activity Processor for an Iterator Activity must stay active but return control to the Processing Sequence controller for each iteration. It does this by calling the ExecuteIteration method provided by the DXACTBAS1 ancestor class for the purpose.

Before calling the ExecuteIteration method it should set any output variables, especially if their values change from one iteration to the next (as would be typical for an Iterator Activity). Refer to 9.6.7 Access the Variable Pool for information on setting output variables values.

Each time your Activity Processor executes the ExecuteIteration method, it returns control temporarily to the Processing Sequence controller, thereby enabling the Processing Sequence controller to execute all the Processing Sequence directives that might be nested beneath the Iterator Activity.

The ExecuteIteration method returns a Boolean value that indicates whether processing should continue. If the return value is False, the Activity Processor must immediately end without attempting to perform further iterations. This may occur, for example, where an exception occurred or a LEAVE Processing Sequence directive was encountered while executing the block of Processing Sequence directives nested beneath the Iterator Activity.

The example below shows how the ActivityRun method is implemented for the supplied FOR\_EACH\_INDEX Activity - a very simple (but nevertheless useful) example of an Iterator Activity.

```
* -----
                                _____
                                                           -----
 * ActivityRun - executes the body of this activity.
 *
                                                                       ------
G-mthroutine name(ActivityRun) options(*redefine)
    define_com class(#dx009v0) name(#llteration)
  pegin_loop using(#gIndexOut) from(#gIndexFrom) to(#gIndexTo) step(#gIndexStep)
      #lIteration += 1
    * restart processing - don't start executing
* iterations until the CurrentIteration value is reached
@-if (#lIteration >= #com_owner.CurrentIteration)
        * put the current index on the scratchpad
#com_owner.PutVariable iname('INDEXOUT') ivalue(#gIndexOut)
       * execute this iteration
—leave if(*Not #com_owner.ExecuteIteration)
     Lendif
   Lend_loop
L
 -endroutine
```

# 9.6.10 Supporting Restartable Activities

If an instance of your Activity Processor ends with an exception return code and if you have defined the Activity with the *Restartable* attribute, then the Processing Sequence run may be eligible to be restarted from the point of failure.

Refer to Activity Details for information about setting the *Restartable* attribute in an Activity definition.

Refer to **Restart a Processing Sequence Run** for information on conditions that affect the eligibility of a Processing Sequence run to be restarted.

Your decision whether to allow your Activity to be *Restartable* will be influenced by a number of factors including:

- The risks of an exception being returned by your Activity Processor that might require a restart
- The need for your business process integration solution to be able to handle the restart in a largely automated manner, given the risks.
- The nature of the logical unit of work that the Activity performs
- The nature of the processing steps necessary to perform that work
- How you choose to implement those steps and whether sufficient information can and will be available through variable data in the variable pool or by any other means to enable your implementation to accurately and reliably restart processing at any given point of failure.

Before making your decision, you should understand that, in general, writing an Activity Processor to be *Restartable* requires greater skill, greater understanding of LANSA Composer processing and is more prone to error.

Remember, too:

- LANSA Composer's ability to restart a Processing Sequence depends on an error status having previously been returned by the activity. (Processing Sequences that end in error due to an abnormal program termination are not restartable.) Therefore, if your Activity Processor does not have any condition that causes it to set an error status in the Activity return code (and it is not an Iterator Activity), there is little point in implementing the restart support.
- Many of the common transport and other Activities supplied with LANSA Composer are supplied as restartable. If you were to consider the risks of

failure in your business process integration solutions, it may be that you would conclude that the risk of failure in the course of normal operation was greater in those areas than in the functions encapsulated by your own Activity Processors. In that case, the costs of implementing this support in your own Activity Processors might not be justified by the benefits.

• The risks of an exception being raised in the course of executing an *Iterator* Activity are multiplied somewhat by the likely or typical number of iterations and the number and nature of items that may be executed by the Processing Sequence controller for each iteration. Because the risks are often greater (and perhaps less certain) for an Iterator Activity, you might more strongly favour implementing restartable support for such an Activity.

If you choose to implement restartable support in your Activity Processor:

- You must implement your solution such that your Activity Processor will have sufficient information available to it from the values of variables in the variable pool or by whatever other means you choose to enable your implementation to accurately and reliably restart processing at any given point of failure.
- You must implement your solution to be capable of reconstructing its state (as far as is necessary) such that it can validly resume processing at the appropriate point.
- If it is an iterator activity, then it must be capable of restarting its processing at the iteration in which the failure occurred.
- The DXACTBAS1 ancestor class has several properties that your Activity Processor can reference to help determine whether restart processing is required and at what point. Some of these are listed below. Use feature help in the Visual LANSA IDE for more information about the individual properties.
  - CurrentIteration
  - Restartable
  - RestartingChild
  - RestartingThis

# 9.6.11 Use the Java Service Manager

The DXACTBAS1 ancestor class implements several methods specifically for loading and calling LANSA Integrator services. The methods provided wrap and simplify the calls to the Java Service Manager (JSM) and perform much of the necessary housekeeping, exception handling and logging.

These methods do, however, assume that you use a single connection to the JSM. If your Activity Processor needs to have multiple connections open, then you will have to write the code to manage the second and subsequent connections yourself.

Your Activity Processor does not have to use LANSA Integrator services. If it does not, you do not need to use these methods.

For more information about developing applications that use LANSA Integrator services, refer to the *LANSA Integrator Guide*.

# ExecuteJSMOpen

Opens a connection to the Java Service Manager for executing LANSA Integrator services. The result is True if successful. Performs exception handling and logging as required. Sets the Activity return code to error if the open fails.

A successful call to ExecuteJSMOpen will usually be followed immediately by a call to the ExecuteJSMLoad method to load the required service.

# ExecuteJSMLoad

Issues the SERVICE\_LOAD service command to load the named JSM service, automatically enabling tracing according to the value specified in the System Settings, and performing exception handling and logging as required. Sets the Activity return code to error if the service load fails.

# ExecuteJSMCommand

Executes a specified service command through the Java Service Manager. Performs exception handling and logging as required. By default, this sets the Activity return code to error if the command fails. However, if the iRecoverable parameter is set to True, the warning level will be set instead.

# ExecuteJSMUnload

Issues the SERVICE\_UNLOAD service command. Performs exception handling and logging as required. Sets the Activity return code to warning if the command fails.

#### ExecuteJSMClose

Closes the connection to the Java Service Manager, optionally unloading the currently loaded service first. Performs exception handling and logging as required.

#### 9.6.12 Signal an Event

Your activity processor may signal an event to the processing sequence controller by using the NotifyEvent method of the ancestor class. This usually used to trigger a notification such as an e-mail when an unrecoverable condition occurs that might require operator intervention. By using an event ID that corresponds to one defined in Event Maintenance you can connect your event with the definition of the action that is to be taken in response to it.

#### 9.6.13 Set the Activity Return Code

When an Activity begins processing its return code is set to 'OK' to indicate a normal completion. If your Activity Processor encounters an exception it should set the return code accordingly by calling one of the following methods in the ancestor class:

- SetCancel (sets the return code to 'CN')
- SetError (sets the return code to 'ER')
- SetWarning (sets the return code to 'WN')

# 9.6.14 Use Logging Services

The DXACTBAS1 ancestor class implements the Logger property. Through the methods of the logger, your Activity Processor can manage its output to the log for the Processing Sequence.

#### **Nesting Level**

The NestAdd and NestSub methods of the logger control the nesting level of the log entries. The nesting level is used as a visual cue when displaying the log entries to indicate the relative hierarchy of the processing steps.

Every call to NestAdd should be matched by a later call to NestSub. It is your responsibility to ensure your program code adheres to this requirement for all possible code paths.

In most cases it is unnecessary for your Activity Processor to call these methods. The Processing Sequence engine manages the nesting level to reflect the structure of the Processing Sequence. However, if the processing in your Activity Processors is particularly structured and you need to reflect that in your log output, you can call these methods.

# Write Log Entries

The Processing Sequence log is held at two levels of detail.

Summary log entries are typically used to indicate completion of major processing steps and are intended for routine operational monitoring. You can create summary log entries as required using the following methods of the logger:

- LogUserComp (completion )
- LogUserDiag (diagnostic)
- LogUserError (error)
- LogUserExtra (subsidiary log text)
- LogUserInfo (information)
- LogUserWarn (warning)

Detail log entries are typically used to provide deeper diagnostic information and are intended for problem diagnosis and resolution. You can create detail log entries as required using the following methods of the logger:

• LogTechComp (completion )

- LogTechDiag (diagnostic)
- LogTechError (error)
- LogTechExtra (subsidiary log text)
- LogTechInfo (information)
- LogTechWarn (warning)

#### **10.** Tips and Techniques for Success with LANSA Composer

This chapter offers some tips and techniques to help you get the most out of LANSA Composer. They are organised in the following sections:

- 10.1 Activities
- 10.2 Processing Sequences

# **10.1 Activities**

The following tips relate to the use of the Activities supplied with LANSA Composer:

• 10.1.1 Experiment with Activities by Running Them

#### **10.1.1 Experiment with Activities by Running Them**

LANSA Composer is supplied with a wide range of Activities. As you explore and discover the Activities available, you might find it useful to simply run an Activity to experiment what it can do.

When working with an Activity in the LANSA Composer client, you can simply run the activity on its own to see what it does. To do this, simply click the Run button to show the Run Activity window:

	Activity	CALCULATE	
	Description	Perform a simple arithmetic calculation	
	O Submit to <u>b</u> atch	Run immediately	
	Type parameter values for this run:		
	First number	22	
	Arithmetic operator (+, -, *, 7, %)	/	
	Second number	7	

The Run Activity window prompts you for the input parameter values. Complete the parameters according to the documentation for the Activity and click Run. LANSA Composer will execute the Activity on your server just as though you had created a Processing Sequence containing the Activity with the parameter values you entered. When the run has completed, LANSA Composer displays the Processing Sequence Log which will show you the input and output parameter values and/or any execution errors that occurred:

- nellesi	🖻 Show detailed	d 🔍 Expand all	💍 Print
	* - 15 dk -	CALCULATE	
<u> </u>	Acuvity	CALCOLATE	
	Description	Perform a simple arithmetic calcul	ation
	Run number	0000000000004406	
	Start date / time	2013-10-20 3:30:41 PM	
	End date / time	2013-10-20 3:30:42 PM	
	Execution result code	ОК	
	Processing started for activity CALCULATE     Log variable value     Log variable value     Starting activity CALCULATE     Activity starting: Perform a simple arithmetic calculation     Calculation successfully completed     Log variable value     Processing ending		LANSA Composer Server version 4.1 (Build 444 DEV) NUMBERIN1 = 22 NUMBERIN2 = 7 OPERATOR = / Starting activity CALCULATE, processor DXACTCALC (type RUP) ation Result: 22 / 7 = 3.142857142 NUMBEROUT = 3.142857142
	Log variat	for activity	Flansed (seconds): 0.127

Clearly not every Activity is suitable to be used in this way, but for many Activities, this can be a quick and easy way to experiment with its capabilities.

#### **10.2 Processing Sequences**

The following tips relate to the design, implementation and management of Processing Sequences in LANSA Composer:

- 10.2.1 Example Processing Sequences
- 10.2.2 Considerations for Extended Duration Processing Sequences

#### **10.2.1 Example Processing Sequences**

LANSA Composer includes a number of example Processing Sequences that illustrate selected features and techniques.

Refer to the following headings for more information:

- Install the Example Processing Sequences
- List of Example Processing Sequences
- Using the Example Processing Sequences

#### **Install the Example Processing Sequences**

If you did not choose to install the examples when prompted during the Server Initialisation Wizard after installing or upgrading your LANSA Composer system, then the examples will not be immediately available. In this case you can locate and import all or selected examples from the supplied DX\_MASTER\_EG.dxexport import file. This file should be present in your LANSA Composer client installation in the DXImports folder, for example:

C:\Program Files\LANSA Composer\X\_WIN95\X\_LANSA\X\_LIC\EXECUT

#### **List of Example Processing Sequences**

The following example Processing Sequences are supplied with this version of LANSA Composer:

Name	Description
EXAMPLE_AATEST1	Check LANSA Composer installation
EXAMPLE_AATEST2	Check LANSA Integrator functionality
EXAMPLE_ACTPARM	Shows ways to provide activity parameter values
EXAMPLE_CSV01	Example of processing CSV
EXAMPLE_EVENT01	Example event notification handler processing seq
EXAMPLE_LOG01	Logging Example
EXAMPLE_LOOP01	Example of processing multiple documents.
EXAMPLE_LOOP02	Example of processing multiple trading partners.

EXAMPLE_PDF01	Example of creating PDF using PDFDocumentService
EXAMPLE_SQL01	Example of using the SQL_QUERY activity
EXAMPLE_SQL02	Example of using the SQL_UPDATE activity
EXAMPLE_SQL03	Example of using the SQL_CALLQRYCSV activity
EXAMPLE_SUSP01	Suspend Example
EXAMPLE_SWIT01	Terminate, loop and conditional example
EXAMPLE_TS01	Example data entry using aXes terminal server
EXAMPLE_WATCH01	Example of using WATCH_DIRECTORY, WAIT_FILESREADY
EXAMPLE_XML01	Example of using the XML_QUERY activity

#### **Using the Example Processing Sequences**

Here are some suggestions on how to get the most out of the example Processing Sequences:

- You can open each of the example Processing Sequences in the Processing Sequence Editor to see what it does and how it is defined;
- You can review the Notes supplied with each Processing Sequence for guidance on its purpose and use;
- You can Run the Processing Sequences on your own system refer to the Notes for any instructions for doing so;
- You can copy and alter the Processing Sequences to experiment further with the techniques they illustrate.

# **10.2.2 Considerations for Extended Duration Processing Sequences**

Many LANSA Composer users implement solutions involving Processing Sequences that run for an extended duration – perhaps continuously.

Frequently such solutions are designed to monitor some source of in-coming data, such as new files arriving in a specified directory, and initiate further processing when an item is detected.

This style of Processing Sequence is very different to one that has a known and finite unit of work to perform, performs it and ends. Consequently, some special considerations apply to the design and implementation of this type of Processing Sequence.

Below we have listed some suggestions that you may wish to consider when implementing such solutions. Refer to the following headings:

- Reduce the logging level
- Allow your processing sequence to frequently enter an idle or wait state
- Delegate transaction processing
- Check for controlled end
- Periodically restart for manageability and performance

The supplied example Processing Sequence EXAMPLE\_WATCH01 is an example of an extended duration processing sequence and implements some of the suggestions in this article. You may wish to examine and run EXAMPLE\_WATCH01 to further investigate the techniques suggested. Refer to 10.2.1 Example Processing Sequences for more information.

# **Reduce the logging level**

You may wish to reduce the logging level for your extended duration Processing Sequences from the values you have specified in System Settings. In selected Processing Sequences you can override the logging values specified in System Settings by changing the values of the *\*process\_loglevel* and *\*process\_jsmtrace* built-in variables. The processing sequence EXAMPLE\_WATCH01, for example, contains the following ASSIGN directives near the beginning of its processing:



Reducing the logging level in this way will reduce the logging burden of your extended duration Processing Sequence without affecting the logging behavior of other Processing Sequences that you run.

If your System Settings specify *Maximum* logging level and *LANSA Integrator tracing* is also selected, then an extended duration Processing Sequence that uses an activity that in turn uses the LANSA Integrator JSM may take a very long time to end after it has completed its normal work. This is usually due to retrieving and logging the large LANSA Integrator trace data that has been generated. You can avoid this by implementing either or both of the logging overrides shown using the built-in variables.

# Allow your processing sequence to frequently enter an idle or wait state

It is usually a characteristic of extended duration processes that they spend a great deal of their time waiting for something to happen – for a file to arrive, for example, or for a message to be received in a message queue.

It is important that your implementation should allow the process to enter an idle or wait state and then to stay in that state for as long as is practical. This will reduce the impact the process has on the system and on other jobs running on the system.

In many cases, activities used in such Processing Sequences have a parameter that specifies how long the activity should wait for the item that it is receiving. Some examples include:

- The DTAWAIT parameter of the DTAQ\_RECEIVE activity;
- The WAITTIME parameter of the MSG\_RECEIVE activity;
- The MSGWAIT parameter of the MSGQ\_RECEIVE activity;
- The WAIT parameter of the WATCH\_DIRECTORY activity;

You should make sure that you set those parameters appropriately. If you choose a value that is too short or, even worse, choose to specify a zero wait, then you may find that your Processing Sequence expends a lot of system resources doing very little but looping endlessly. Using a sensible wait value will allow the Processing Sequence to frequently enter an idle/wait state when it has no work to do.
Note that, at the other extreme, specifying an indefinite wait is not always a good idea either. Refer to Check for controlled end to find out one reason why.

If your extended duration Processing Sequence uses an activity, such as MAIL\_RECEIVE or FTP\_INBOUND for example, that does not have a built-in "wait" function and you are using that activity to "poll" for in-coming data, then you should consider combining the activity in a loop with the SLEEP activity to create a similar effect to, for example, the WAIT parameter of the WATCH\_DIRECTORY activity.

#### **Delegate transaction processing**

Frequently extended duration Processing Sequences are designed to monitor some source of in-coming data, such as new files arriving in a specified directory, and initiate further processing when an item is detected.

It is usually good practice to separate the former (the waiting and discovery) from the latter (the processing). The less work that your extended duration Processing Sequence does, the better. As soon as it "discovers" a new item to process, it should usually initiate another *asynchronous* process to do the processing, while it stays busy with waiting for further new items.

This approach has the additional benefit that the processing of any individual transaction does not create a bottleneck that prevents or delays the receipt and processing of further transactions.

You can use the **COMPOSER\_RUN** activity in your extended duration Processing Sequence to initiate the secondary Processing Sequence through the LANSA Composer request server. Remember to specify 'NO' for the SYNCHRONOUS parameter to make sure that the first Processing Sequence does not have to wait for the secondary Processing Sequence to complete.

## Check for controlled end

Frequently, it is the very nature of an extended duration Processing Sequence that its task is infinite, or at least indefinite. That is, it will continue monitoring for whatever it is that it monitors throughout the business day and perhaps the night and weekends too.

Activities such as WATCH\_DIRECTORY are specifically designed for this indefinite processing. This activity will continue iterating until or unless your solution ends the loop explicitly or until the job, subsystem or system end.

Alternatively, if your extended duration Processing Sequence uses an activity,

such as MAIL\_RECEIVE or FTP\_INBOUND in a loop to monitor for new items, you have probably designed the loop to be indefinite using a LOOP, WHILE or UNTIL directive.

In either event, it is good practice to implement the solution such that a controlled end for it can be externally triggered. In most cases, this can be done by checking and acting upon the value of the *\*shutdown* built-in variable at an appropriate point in your solution. The processing sequence EXAMPLE\_WATCH01, for example, implements such a provision using the condition specified on a LEAVE directive:



This will allow you to end the processing Sequence run using the *Processing Sequence Log* window in the LANSA Composer client or using the *Operations Console*. The *Processing Sequence Log* window, for example, provides an *End* toolbar button or the *End* command on the *File* menu – these are shown only when the Processing Sequence run is *active*.



If controlled end for a Processing Sequence run is requested in this way, then the condition shown above will be satisfied. In the case of EXAMPLE\_WATCH01, this will result in the main WATCH\_DIRECTORY iterator loop ending and the Processing Sequence will then end too.

On IBM i servers, the test of the \*shutdown built-in variable will also detect when a controlled end has been initiated for the IBM i job, subsystem or system, using the ENDJOB, ENDSBS or PWRDWNSYS commands or equivalents.

By placing a test for controlled shutdown in an appropriate place in your extended duration Processing Sequence in this way, you can help to ensure that, when the Processing Sequence is shutdown, it is done in a controlled fashion and at an appropriate transaction boundary, thereby protecting the data integrity of your solution and your application.

#### Periodically restart for manageability and performance

As already discussed, it is frequently in the very nature of an extended duration Processing Sequence that its task is infinite, or at least indefinite. While that may be so, there are several very good reasons that you might not want the task to be served with a single Processing Sequence run that runs over many hours or even days. These reasons include:

- The Processing Sequence Log will become exceptionally large, which will make it slow to load and exceptionally difficult to analyse in the event that you have to perform troubleshooting;
- Over an extended time, and for various reasons, the system resources consumed by the job containing the Processing Sequence run may gradually increase, possibly impacting performance on the system as a whole;

It is usually good practice to limit the lifetime of a single Processing Sequence run, allow it to end and begin another to resume and continue its processing. Doing so is likely to offer benefits with respect both to the manageability and the performance of the solution.

There is more than one way to accomplish this. The Processing Sequence EXAMPLE\_WATCH01 implements the following sequence of directives at an appropriate point inside its main WATCH\_DIRECTORY iterator loop for this purpose:



The COMPOSER\_RUN activity simply submits the same Processing Sequence again through the LANSA Composer Request Server, and then the Processing Sequence ends. In this instance, this occurs after 999 iterations (the value of the variable &ITERATIONS\_MAX is set to 999 earlier in the Processing Sequence).

This results in multiple Processing Sequence runs, each covering a specific time period, which makes it much easier to examine the logs and diagnose issues, when necessary.

### LANSA Composer Tutorials

The LANSA Composer tutorials are designed to introduce the basic elements of the LANSA Composer application.

This section covers some important topics:

How do I use the Tutorials?

LANSA Composer Server objects required for the Tutorials

**Resetting Tutorial Data** 

#### Before you begin

In the first six exercises of the tutorials you will create a Composer Processing Sequence that will perform the following steps for each Trading Partner:

- Find xml files in the "Inbound Tutorial" directory linked to the Trading Partner
- Read each xml file and update a database table with its data
- Send an acknowledgement email to the trading partner.

The following tutorials are included:

LIC001 - Introduce Composer Client & Validate Environment

- LIC002 Create a Processing Sequence
- LIC003 Create a Transformation Map
- LIC004 Add the Transformation to the Processing Sequence
- LIC005 Set up a Trading Partner
- LIC006 Add Email Acknowledgement to Processing Sequence
- LIC007 Extract Database to CSV File
- LIC008 Processing via Email
- LIC009 Handle multiple requests via an email attachment
- LIC010 Calling a Processing Sequence (Optional)
- LIC011 Email notification service

#### How do I use the Tutorials?

It is recommended that you complete the tutorials in sequence. The objects created in the earlier tutorials are reused and extended in the following tutorials. If the tutorials have been used previously, you should reset the tutorial data, as described in Resetting Tutorial Data.

Some sections of the tutorials will vary according to the server you are using. The examples given in these exercises are for an IBM i server. If you are using a Windows server you will have to use appropriate values for a Windows server. The sections which will vary are:

- Setting up an ODBC connection within Altova MapForce for the database connection.
- Setting up a Database connection identifier. This is defining a JDBC connection on your server.
- Ensuring the database tables used in your Transformation are under commitment control.

### LANSA Composer Server objects required for the Tutorials

The objects required for the tutorials are installed with the LANSA Composer server software.

If any of the tutorial objects have been damaged or deleted you can find them:

- For the IBM i server, in save file DXMASTIFS in your partition data library (for example, **LICLICLIB**).
- For the Windows server, in file tutorial.zip in your partition execute directory

for example: c)

The tutorial objects you need are listed below:

• Database tables: TUTORDH TUTORDL

For the IBM i server, the database tables are supplied and installed as \*FILE objects in your partition data library (for example, LICLICLIB).

- For the Windows server, the database tables are supplied and installed in a Microsoft Access database, TUTORIAL.mdb, in your partition execute directory: for example, C:\Program Files\LANSA\X WIN95\X LANSA\X LIC\Execute
- Flat files installed into a directory on your server: TUTorder.dtd TUTorder.xml Ack.txt

TUT\_01\_SKEL.txt TUTORIAL.mdb (Windows server only)

- Activities: TUT\_01\_AT TUT\_02\_AT
  - Processing Sequence: TUT\_01\_P1 TUT\_01\_P2 FSTUTSEQ (example of the completed Processing Sequence)
- System setting

The *Tutorial Files* system setting should be set to the directory location where the flat files are installed.

• LANSA system variable \*AUTOALP10BATCHNO

## **Resetting Tutorial Data**

If the tutorials have been used previously, it is recommended that you clear the previous results as follows:

- Delete any data in tables **TUTORDH** and **TUTORDL**. These tables will exist on your server.
- If you are using an IBM i server they will exist in the library being used for the LANSA partition where Composer was installed (for example, LICLICLIB).
- If you are using a Windows server, they will exist in a Microsoft Access database, TUTORIAL.mdb, in your partition execute directory, for example, C:\Program Files\LANSA\X\_WIN95\X\_LANSA\X\_LIC\Execute

Delete from the tutorial directory flat files:

iii\_TUT\_01\_P1.txt iii\_TUT\_01\_P2.txt

**Note:** There may be several of these files where iii represents the initials entered by a person who has previously used the tutorials.

• Do NOT delete TUT\_01\_SKEL.txt

For exercises 7, 8, and 9 you will also require the LANSA Personnel Demonstration system installed

### Before you begin

In order to complete these tutorials, you must have completed the following:

- Installed and configured Composer on your server.
- Installed and configured Composer Client on your Microsoft Windows PC.
- Obtained a valid server user ID and password to use with Composer.
- Installed and configured the base LANSA Integrator on your server.
- Install Altova MapForce and have a valid MapForce license.
- Installed the LANSA Personnel Demonstration system

It is recommended that you complete the tutorials that are provided with MapForce to familiarize yourself with the mapping environment.

## 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

#### LIC001 - Introduce Composer Client & Validate Environment

#### Objectives

- To start the Composer Client and connect to your server.
- To execute a supplied Composer Processing sequence to validate your installation.
- To execute a supplied Composer Processing sequence which uses LANSA Integrator to validate the LANSA Integrator installation.

#### Steps

To achieve these objectives, you will complete the following:

- Step 1. Locate and execute Composer Client
- Step 2. Execute a Composer Processing sequence to validate your installation
- Step 3. Execute a Composer Processing sequence to validate the base LANSA Integrator installation
- Summary

#### Step 1. Locate and execute Composer Client

1. Locate the Composer client:

When Composer was installed on your PC, it created a program group named *LANSA Composer* with a number of icons in it. On your desktop or under programs on your *Start* menu, locate and open the program group named LANSA Composer.

2. To start the Composer client:

In the LANSA Composer program group, click or double-click the *LANSA Composer* icon to open the *Connect to LANSA Composer server* dialog.

Connect to LANSA Composer s	erver	Connect	
Composer	Cancel		
		Work <u>O</u> ffline	
Type the user name and password to co	onnect to your LANS	A Composer server.	
User	LICPGMLIB		
Password	****		
Type or select the LANSA Composer ser	ver to which you wi	sh to connect.	*
Server connection name	COMPOSER01		-
Partition	LIC		
Click to specify connection attributes fo	r your LANSA Compo	oser server.	*

- 3. Enter the *User* name and *password* that you will be using. Select the Composer application server that you want to connect to and then press the *Connect* button. For details of the log on procedure, refer to Connect to LANSA Composer Server.
- 4. The LANSA Composer main window will be displayed.

Familiarize yourself with the options that are available in the *Navigator* on the left.

# **Step 2. Execute a Composer Processing sequence to validate your installation**

- 1. Expand *Definitions* in the *Navigator*, then select *Processing sequences*.
- 2. Locate the *Processing Sequence* with identity TUT\_01\_P1. You may search for it with a partial ID (such as TUT) or blank (to view all IDs). Select *Processing Sequence* "**TUT\_01\_P1 Tutorial 01 check environment**".



You may view the contents of this *Processing Sequence* by selecting the *Edit* button. When you do so, you will see a message box, informing you that this is a supplied Composer processing sequence, and may be opened read only. Close the edit window.

- 3. To execute the *Processing Sequence*, click on the *Processing*
- 4. The *Run Processing Sequence* window is shown. Select the *Run immediately* radio button to select it. Type your own values for the *initials, message* and *name* parameters.

🕨 Run Proc	essing Sequence		×
1	Processing sequence Description	TUT_01_P1 Tutorial 01 - Check environment	
	O Submit to <u>b</u> atch	Run immediately	
	Type parameter values for this run:		
	Enter your initials	'FS'	
	Enter your message	'Hello World'	
	Enter your name	'Fred Smith'	
200000000000000000000000000000000000000	R <u>u</u> n C	ancel 🔽 Close	err

5. Click on the *Run* button at the bottom of this window.

When the *Processing Sequence* completes, the steps and results of the processing are shown in the *Processing Sequence Log* window.

1	Processing sequence	TUT_01_P1	
1	Description	Tutorial 01 - Check environment	
	Run number	00000000001202	
	Start date / time	12/01/2009 11:26:51 AM	
	End date / time	12/01/2009 11:26:52 AM	
	Execution result code	OK 💌	
	<ul> <li>Parameter value</li> <li>Parameter value</li> <li>Statting activity</li> <li>Activity start</li> <li>Your message</li> <li>Starting activity</li> <li>Activity start</li> <li>Processing sequence</li> </ul>	TUT_01_AT ing:Tutorial 01A Activity ge file is TEXT_SUBSTITUTE ing:Substitute into Text skeleton ie ended	YOURMESSAGE = Hello World YOURNAME = Fred Smith Starting activity TUT_01_AT, processor DXACTTUT1 (type RUP) /LANSA_Composer2/Tutorial/FS_TUT_01_P1.txt Starting activity TEXT_SUBSTITUTE, processor DXACTTX02 (typ. Elapsed (seconds): 0.117
	2		

Alternatively, you may view the complete log information by selecting *Console* in the *Navigator* (under *Operations*). Locate your processing sequence run using the *Processing Sequence ID* (TUT\_01\_P1) and date and time. To view the log information for the execution of this Processing Sequence, click the magnifying glass icon under the *Run Number* heading.

6. If successful, this *Processing Sequence* took the input values entered by you

and merged them into a text skeleton to create a text file. The text skeleton is TUT\_01\_SKEL.txt. It is located in the Tutorial directory on your server. The text file created is **iii\_TUT\_01\_P1.txt** in the tutorial directory (where iii are the initials you entered). If you do not know what the tutorial directory is, it is recorded in the *Tutorial Files* system setting. To display the value of this system setting, expand *Administration and Housekeeping* in the *Navigator*, and select *System settings*.



Locate this directory (using Windows explorer or similar) and view the contents of the file *iii\_TUT\_01\_P1.txt* (where iii is the initials you entered). On an IBM i server, you may need to log on to the server.

Your system administrator can supply the IBM i domain name and your IBM i User ID and Password.

This could be your initial view:

😂 OS/400 (earth)			
File Edit View Favorites Tools	Help		
🚱 Back 🔹 🕥 - 🎓 🔎 Sea	rch 😥 Folders	× ♥	
Address 🖳 Mearth			💌 🄁 Go
Folders ×	Name 🔺	Comments	
Comparison of the second	CP3WEBDOCS D11HTDOCS D11JSM D11PGMLIB	D11PGMLIB htDocs	
	C@JSM C@WEBDOCS EDIDIRECT	DC@PGMLIB J5M webdocs for dc@pgmlib	
🗐 WinV10.zip 🗉 🛃 Control Panel	SINTRANET	intranet,lansa.co.uk	
Mobile Device     Shared Documents      Gan Ban's Documents	QDIRSRV QDL5 OIBM	OS/400 Directory Services Shared Folders IRM Product Directories	
Danny's Documents      Doff's Documents	R10PGMLIB SLSPGMLIB		
John2's Documents      Geometry     My Sharing Folders      Mokia Phone Browser	T11HTDOCS	sis copy log hies T11PGMLIB HTDocs T11PGMLIB Images	
Sue's Documents     Sue's Network Places     Sue's Entire Network	T11JSMLIB TRNHTDOCS	T11JSMLIB Training JSM	

The paths shown here will depend on how your network access to the IBM i IFS (or Windows server computer) has been defined. Some of the above paths

may be aliases to IFS folders, for example, D11PGMLIB as in this example. Others are actual IFS folders, such as QDLS.

The following screen picture shows the path you are looking for – in this example as defined on the IBM i server "EARTH", with LANSA program library "D11PGMLIB".

😂 Tutorial				
File Edit View Favorites Tools	Help			<b>.</b>
🚱 Back 🔹 🌍 - 🏂 🔎 Se	arch 😥 Folders 🕼 🍞 🗙 🗳	•		
Address 🛅 \\earth\D11PGMLIB\LANSA_C	omposer\Tutorial			💌 🄁 Go
Folders ×	Name 🔺	Size Type	Date Modified	
🗄 🧰 Ben's Documents 🛛 🔺	🖬 810.edi	1 KB EDI File	30/10/2007 09:35	
Danny's Documents	🗐 Ack.txt	1 KB Text Document	10/05/2007 23:54	
Joff's Documents	F5 x_TUT_01_P1.txt	1 KB Text Document	10/10/2007 10:03	
E 🛅 John2's Documents	F5_TUT_01_P2.txt	1 KB Text Document	10/10/2007 10:05	
🗉 🙀 My Sharing Folders	JMI_TUT_01_P1.txt	1 KB Text Document	31/10/2007 16:12	
🗉 🖬 Nokia Phone Browser	JMI_TUT_01_P2.txt	1 KB Text Document	31/10/2007 16:14	
🗉 🫅 Sue's Documents	Bog JMI_Tutorial.mfd	6 KB Altova MapForce D	30/10/2007 17:24	
🖃 📢 My Network Places	TUT_01_SKEL.txt	1 KB Text Document	10/05/2007 23:55	
🖃 🎑 Entire Network	TUTorder.dtd	1 KB DTD File	12/10/2000 04:48	
🗉 🚯 Microsoft Terminal Service:	TUTorder.xml	1 KB XML Document	15/01/2007 07:32	
🖃 👩 Microsoft Windows Networ				
🗉 💏 Lansauk				

The existence of this file with your details validates that you can execute a *Processing Sequence* using *LANSA Composer*.

📮 JMI_TUT_01_P1.txt - Notepad	×
File Edit Format View Help	
Welcome <your, name=""> to LANSA Composer.</your,>	^
Your message is : Hello Composer	
	Y

# **Step 3. Execute a Composer Processing sequence to validate the base LANSA Integrator installation**

To execute this step you must have the base *LANSA Integrator* installed, configured and running on your server.

1. Select the *Processing Sequence* TUT\_01\_P2.

cons	By ID By Status	🚱 Search	Sequence ID A TUT_01_P1 TUT_01_P2	Description Tutorial 01 - Check environment Tutorial 01 - check JSM	Status t Active Active
	Processing sequence	: Details (TUT_0 nds   🔃 Run histo TUT_01_P2	1_P2-Tutorial O ry   🖶 Version I	1 - check JSM) nistory   🚺 Notes   🐼 Audit	
ations	Description Restartable	Tutorial 01 - check Yes	JSM		Refresh
)ns ns	Status	Active	•		Run
keepir	Click Edi	t to edit your proces	ising sequence.		Edi <u>t</u>

- 2. Click the Run... button.
- 3. The *Run Processing Sequence* window is shown. Select the *Run immediately* radio button. Type your own values for the *initials* and *directory* parameters.

🕨 Run Proc	cessing Sequence		
1	Processing sequence Description	TUT_01_P2 Tutorial 01 - check JSM	
	Submit to <u>b</u> atch	Run immediately	
	Type parameter values for	this run:	
	Enter your initials	'FS'	
	Directory to be listed		
	R <u>un</u> C	ancel 🔽 Close	8

For the *directory* parameter, you should type the full path of a directory on your server as defined in the system settings/server file locations (for example, for IBM i '/LANSA\_Composer\_licpgmlib/lic/Tutorial' or for Windows C:\Composer\lic\Tutorial).

Refer to your system administrator for the requisite directory path set up for the training environment.



*Reminder:* If you are using an IBM i server the directory separator should be '/'. If you are using a Windows server the separator should be '\'.

4. Click the *Run* button. When the *Processing Sequence* completes, the steps and results of the processing are shown in the *Processing Sequence Log* window.

If successful, this *Processing Sequence* listed the files contained in the specified directory. This used the *LANSA Integrator* file client service. The

list of files was read by the second *Activity* in the *Processing Sequence*, which reported the first five files paths and names in a text file. This text file is named iii\_TUT\_01\_P2.txt and can be found in the tutorial directory (where iii is the initials you entered).

5. To review the results. As above, navigate to the tutorial directory (using Windows explorer or similar) and view the contents of the file iii\_TUT\_01\_P2.txt (where iii is the initials you entered).

JMI_TUT_01_P2.txt - Notepad	
File Edit Format View Help	
First five entries in Directory /LANSA_D11PGMLIB/LANSA_Composer/Tutorial No File extension filter. /LANSA_d11pgmlib/LANSA_Composer/Tutorial/Ack.txt /LANSA_d11pgmlib/LANSA_Composer/Tutorial/TUTorder.dtd /LANSA_d11pgmlib/LANSA_Composer/Tutorial/TUTorder.xml /LANSA_d11pgmlib/LANSA_Composer/Tutorial/TUTorder.xml /LANSA_d11pgmlib/LANSA_Composer/Tutorial/FS x_TUT_01_P1.txt	<

#### Summary

- Notice that there can be multiple servers listed in the *Log on* window. This allows *Composer Client* to work with multiple *Composer Servers*. For example, you may have a production server on which you execute live processing. You may also have a development server on which you develop and test new processing.
- The successful execution of the two supplied *Processing Sequences* validates your installation.

### LIC002 - Create a Processing Sequence

#### Objective

• To introduce you to creating your own *Processing Sequences*. The *Processing Sequence* will initially find xml files in the tutorial directory. This sequence will be extended in following tutorials.

#### **Before You Begin**

In order to complete this tutorial, you should have completed the following:

- LIC001 Introduce Composer Client & Validate Environment
- You may wish to refer to *Edit Processing Sequence Details* in the *Composer User Guide* for a description of the controls available to you.

#### Steps

To achieve these objectives, you will complete the following:

- Step 1. Create a Processing Sequence
- Step 2. Configure the Processing Sequence Editor
- Step 3. Add Directory List Activity to your Processing Sequence
- Step 4. Add TUT\_02\_AT Activity to Processing Sequence
- Step 5. Execute the Processing Sequence you Created
- Summary

#### **Step 1. Create a Processing Sequence**

1. Under *Definitions* in the *Navigator*, select *Processing sequences*.



2. Select *low icon* from the tool bar or right click and select *New* to launch the *Processing Sequence* editor in a new window.



#### **Step 2. Configure the Processing Sequence Editor**

The appearance of the Processing Sequence editor can be configured from the *View / Options* menu:



The options at the top of the *Options* dialog can be selected to change the appearance of the editor as shown:

Processing Sequence Editor Options	"The design of the local division of the loc	-	x
Choose the view options to apply in the Proces the effect of the options on the sample below.	sing Sequence Editor. Review	OK Cancel	
Show tree lines			
Show sequence numbers		Reset	
	)		
Cobur code according to nesting level			
Number of colours			
	5 6 Z		
Sample:			
Item	Details	Seq	*
8/1	Nest level 1		
	Next level 2	-	
🗆 📣 Case	most never a	4	
Condition:	8A = 8B		=
1.	Nest level 3		
⊕-{ <sup>1</sup> / <sub>1</sub> } If		6	
Condition:	8/C = 8/D		
Ci lana	nest level 4		
List:	& IST	•	
-1 *	Nest level 5		
🗆 🍎 While		10	
Condition:	8E < 8F		
1.	Nest level 6		
🕀 🚺 Until		12	
Condition:	8kG = 10		
Therefore	nest level 7		
Name:	FOR EACH INDEX - Iterate for	14	
There are a second se	. engener geneen melder fen		

1. Choose the options to suit your preference. All examples shown in this tutorial will have these options turned off.

#### Step 3. Add Directory List Activity to your Processing Sequence

1. Select the *Activities* tab (top left) to display a list of Activities. The list may be filtered by selecting an *Activity group* from the drop down list.

Note that Activities are shown in groups.

 Locate the *File Management* group and then locate the DIRECTORY\_LIST *Activity* and drag and drop it on to the sequence details panel (top right). Drop it on the \*\*\* End of processing sequence \*\*\* text. Alternatively, you may add the *Activity* to the sequence by double clicking on it.



The DIRECTORY\_LIST Activity is a supplied *Activity*. It will output a list of files in a requested directory. Optionally, you can filter the list of files to select only files with a particular extension.

3. Select the *DIRECTORY\_LIST* Activity in the *Details* panel (top right). The details for the *Activity* will be shown in the panel bottom right.

tem	1		Details			
			*** Beginning of p	rocessing sequence ***		
1	<b>}</b>	Activity				
	ſ	Name:	DIRECTORY_LIST **** End of process	- List contents of a dir sing sequence ***	ectory	
eta	ails	Parameter	s Variable or value for paramet	er Parameter name	Parameter description	Parameter clas
eta	ails *	Parameter	s Variable or value for paramet	er Parameter name DIRECTORY	Parameter description Directory to be listed	Parameter clas
eta	ails *	Parameter Inbound Inbound	s Variable or value for paramet '*FILES'	er Parameter name DIRECTORY CONTENT	Parameter description Directory to be listed Content to be listed (*FILES, *DIRE	Parameter cla: Folder path General
eta 2	ails *	Parameter Inbound Inbound Inbound	S Variable or value for paramet '*FILES' 1	er Parameter name DIRECTORY CONTENT MAXDEPTH	Parameter description Directory to be listed Content to be listed (*FILES, *DIRE Maximum depth to be listed (1-n, *)	Parameter cla Folder path General General
eta 1	ails *	Parameter Inbound Inbound Inbound Inbound	S Variable or value for paramet '*FILES' 1	er Parameter name DIRECTORY CONTENT MAXDEPTH FILENAME	Parameter description Directory to be listed Content to be listed (*FILES, *DIRE Maximum depth to be listed (1-n, * File name mask (eg: abc*.txt)	Parameter cla: Folder path General General General
eta 1 2 3	ails *	Parameter Inbound Inbound Inbound Inbound Inbound	S Variable or value for paramet '*FILES' 1	er Parameter name DIRECTORY CONTENT MAXDEPTH FILENAME EXTENSION	Parameter description Directory to be listed Content to be listed (*FILES, *DIRE Maximum depth to be listed (1-n, * File name mask (eg: abc*.txt) File extension filter (eg: xml)	Parameter clas Folder path General General General General
eta 1 2 3 4 5	ails *	Parameter Inbound Inbound Inbound Inbound Inbound Inbound	Variable or value for paramet	er Parameter name DIRECTORY CONTENT MAXDEPTH FILENAME EXTENSION ORDERBY	Parameter description Directory to be listed Content to be listed (*FILES, *DIRE Maximum depth to be listed (1-n, * File name mask (eg: abc*.txt) File extension filter (eg: xml) Order the list by (*NONE *NAME	Parameter class Folder path General General General General General

4. Select the *Parameters* tab (bottom right) to view the parameters for this *Activity*. The *Notes* relating to this *Activity* are shown in the bottom left panel.

🕢 Assistant 😽 Errors	Det	ails	Parameter	s			
				Variable or value for par	Parameter n	Parameter description	Parameter cl.
	1	*	Inbound		DIRECTORY	Directory to be listed	Folder path
This activity will list the contents of a directory.	2		Inbound	'*FILES'	CONTENT	Content to be listed (*FILES,	General
directories contained in the specified directory.	3		Inbound	1	MAXDEPTH	Maximum depth to be listed (	General
You can also specify whether the list should include	4		Inbound		FILENAME	File name mask (eg: abc*.txt	General
specified directory.	5		Inbound	Q2	EXTENSION	File extension filter (eg: xml)	General
Optionally, files (but not sub-directories) may be	6		Inbound	'*NONE'	ORDERBY	Order the list by (*NONE *NA	General
filtered using the FILENAME and/or EXTENSION	7		Outbound		FILELIST	List of files or directories	File path

5. For this tutorial you will select only files with an extension of xml. So, for the parameter *EXTENSION* enter a value of **xml** in the *Variable or value for parameter* column. This means that when this Processing Sequence is executed a value of *xml* will be set for the *EXTENSION* parameter.

		THE REPORT OF A DESCRIPTION OF A DESCRIP						
tem	n		Details			F		
			*** Beginning of pro	cessing sequen	ice ***			
	<b>}</b>	Activity						
Name:		Name:	DIRECTORY_LIST -	DIRECTORY_LIST - List contents of a directory				
Det	ails	Parameter	\$					
Det	ails	Parameter	s Variable or value f	Parameter n	Parameter description	Parameter cl.		
Det	ails	Parameter	s Variable or value f	Parameter n DIRECTORY	Parameter description Directory to be listed	Parameter cl. Folder path		
Det 1 2	ails	Parameter Inbound Inbound	s Variable or value f '*FILES'	Parameter n DIRECTORY CONTENT	Parameter description Directory to be listed Content to be listed (*FILES,	Parameter cl. Folder path General		
Det 1 2 3	ails	Parameter Inbound Inbound Inbound	S Variable or value f '*FILES' 1	Parameter n DIRECTORY CONTENT MAXDEPTH	Parameter description Directory to be listed Content to be listed (*FILES, Maximum depth to be listed (	Parameter cl. Folder path General General		
Det 1 2 3 4	ails	Parameter Inbound Inbound Inbound	S Variable or value f '*FILES' 1	Parameter n DIRECTORY CONTENT MAXDEPTH FILENAME	Parameter description Directory to be listed Content to be listed (*FILES, Maximum depth to be listed ( File name mask (eg: abc*.txť;	Parameter cl. Folder path General General General		
Det 1 2 3 4 5	ails *	Parameter Inbound Inbound Inbound Inbound	S Variable or value f '*FILES' 1 xml	Parameter n DIRECTORY CONTENT MAXDEPTH FILENAME EXTENSION	Parameter description Directory to be listed Content to be listed (*FILES, Maximum depth to be listed ( File name mask (eg: abc*.txt File extension filter (eg: xml)	Parameter cl. Folder path General General General General		
Det 1 2 3 4 5 6	ails	Parameters Inbound Inbound Inbound Inbound Inbound	S Variable or value f '*FILES' 1 xml '*WONE'	Parameter n DIRECTORY CONTENT MAXDEPTH FILENAME EXTENSION ORDERBY	Parameter description Directory to be listed Content to be listed (*FILES, Maximum depth to be listed ( File name mask (eg: abc*.txť; File extension filter (eg: xml) Order the list by (*NONE *NA	Parameter cl. Folder path General General General General General		

6. When this *Processing Sequence* is executed the value for the *DIRECTORY* parameter is going to be provided. This parameter needs to be identified as an input parameter. To do this select the *Variables* tab (top left). The *Variables* tab lists all variables defined in this *Processing Sequence* and all those required by any of its processing steps.

Select the *Parameters* tab (top right) and drag and drop the *DIRECTORY* parameter from the *Variables* tab into the *Parameters* tab.



Note:

You have just defined a runtime parameter for the *Processing Sequence*.

You can define a default value for parameters. This is a useful if a path and file name is being input, to avoid errors while testing.

## Step 4. Add TUT\_02\_AT Activity to Processing Sequence

In this step you will make this processing sequence produce some output, by using the supplied tutorial activity TUT\_02\_AT which was used in the previous exercise.

The activity TUT\_02\_AT accepts two input parameters:

- YOURINITIALS used to prefix the output text file name
- FILELIST the name of the list of files produced by the *Directory List* activity. This defaults to &FILELIST the outbound parameter from the *Directory List* activity.

TUT\_02\_AT has one outbound parameter – the output file name. This defaults to iii\_TUT\_01\_P02.txt where iii is the input "initials" parameter value.

The text produced by TUT\_02\_AT is hardcoded (except for the list of files). In a later module you will see how this type of processing can use an input text file.

1. Select the *Activities* tab (left hand side) and expand the *Tutorials* group.Drag and drop the activity TUT\_02\_AT to the end of the processing sequence.



**Note:** The example shown above has used the search icon to locate Activities beginning "TUT".

- 2. In this step you will add the YOURINITIALS variable as an input parameter to the processing sequence.
  - a. Select the *Variables* tab (left hand side). On the right hand side select the *Parameters* tab.
  - b. Drag and drop the variable YOURINITIALS onto the *Parameters* tab.

Save 💾 🛛 Run 🕨 🝦 💥 🔏 🗊	6	🤣 Ena	abled 🧔	Promote 😭 😽	Demote 🖒	
🥵 Palette   🍰 Activities	🤗 Palette 🊔 Activities 📳 Details 📥 Parameters					
Processing sequences				Parameter Name	Parameter description	Default value
🗟 Transformations 🛛 谷 Configurations	1	Inbound	Required	DIRECTORY	Directory to be listed	
Variables In Ruit-ins	2	Inbound	Required	YOURINITIALS	Enter your initials	FS
⊕ DIRECTORY     € EXTENSION     ♥ EXTENSION     ♥ FILELIST     ♥ ♥ FILELIST     ♥ ♥ FILENAME     ♥ ♥ TEXTFILE     ♥ ♥ YOURINITIALS						

You have now defined two runtime parameters for this processing sequence.

3. Save the *Processing Sequence* by clicking on the Save 🕒 *Save* icon in the toolbar. A window requesting details of your Processing Sequence will appear.

Save		
	Type name and desc	ription for new processing sequence:
	Name:	JMITUTSEQ01
	Description:	JMI Tutorial - LIC002
	Status:	Active
	Save	Cancel

- a. Enter a *Name*, iiiTUTSEQ01. (where iii are your initials).
- b. Enter a *Description*. For example **Fred Smith LIC tutorial LIC002**.
- c. For *Status* select *Active* from the drop down list.
- 4. Click the *Save* button.
- 5. Close the *Processing Sequence Editor*.

#### Step 5. Execute the Processing Sequence you Created

1. Locate and select your *Processing Sequence*. (You may need to search for it with a full or partial ID - such as your initials (*iii*), if you have used the suggested names)



- 2. To execute the *Processing Sequence*, click on the button.
- 3. The *Run Processing Sequence* window is shown. Select the *Run immediately* radio button. Complete the parameters as follows:
  - a. For the *DIRECTORY* parameter, enter the path to the LANSA Composer Tutorial as used in previous exercises.
  - b. Enter your initials

Þ Run Proc	essing Sequence		
1	Processing sequence Description	IIITUTSEQ01 III Tutorial - LIC002	
	Submit to <u>b</u> atch	Run immediately	
	Type parameter values for	this run:	
	Directory to be listed	/LANSA_composer_k12pgmlib/lic/tutorial	
	Enter your initials	JWL.	
Close	R <u>un</u> C	ancel <u>S</u> ave shortcut	

4. Click on the *Run* button at the bottom of this window.

When the *Processing Sequence* completes, the steps and results of the processing are shown in the *Processing Sequence Log* window. Verify that your processing sequence ended successfully by reviewing the log.

5. Locate the tutorial directory on the server (using Windows explorer or similar) and view the contents of the file *iii\_TUT\_01\_P2.txt* (where iii is the initials you entered). Your file should look like the following. Note that because you added a filter ('xml') to the Directory List activity, only one file in the directory has been listed.



You have completed this exercise.

#### Summary

- You have created a *Processing Sequence* using two supplied activities. This is where you can set up your own execution structures using pre-defined building blocks and your own tailored processing entities.
- The *Processing Sequence* you have created will be extended in the following tutorials to introduce you to additional features within the Processing Sequence structure.
- You have viewed the output text file produced by activity TUT\_02\_P02
- You have viewed the *Processing Log* which gives you information on the steps and result of the processing.

## LIC003 - Create a Transformation Map

#### Objectives

- To introduce you to creating a *Transformation Map*. The map will input an xml file and insert the data into a database table.
- To prepare the map for use within Composer
- To set up the database Configurations for the map so the database may be accessed at execution time by the Processing Sequence.

## **Before You Begin**

In order to complete this tutorial, you should have completed the following steps:

- LIC001 Introduce Composer Client & Validate Environment
- LIC002 Create a Processing Sequence

## Steps

To achieve these objectives, you will complete the following:

- Step 1. Create a Transformation Map Registration
- Step 2. Edit the Transformation Map
- Step 3. Identify the xml source for the Transformation Map
- Step 4. Identify the Target Database for the Transformation Map
- Step 5. Map the Input xml to the Target Database
- Step 6. Prepare your Transformation Map for use within Composer
- Step 7. Set up the Database Configuration
- Summary

## **Check System Settings**

Before you attempt to create and use a *Transformation Map*, you should check that the values of the following system settings are appropriate for your environment:

- *Server Network path*: this value defines the prefix that the LANSA Composer client software uses to access files on the server. It is combined with the value specified for *Map definitions* and so must be correct for your environment in order for the *Map definitions* setting to be effective.
- *Map definitions*: this should specify the location on your server where

Transformation Map definitions will be stored and accessible for all LANSA Composer users

- *Map generate and compile*:. This specifies a file system location where intermediate files will be created when the maps are prepared. Usually the default behavior (to use the Windows temporary directory for the user) is acceptable.
- *LANSA Integrator (JSM) instance*: this specifies the path from the client to the JSM Instance folder on the server. This controls where the compiled Java archive file for a *Transformation Map* is placed when the *Transformation Map* is *Prepared* (it is placed in the \composer\LIC\map sub-folder of the path specified here, where LIC is the LANSA partition).

🙍 System settings				
System Settings	Map definitions-			
🖃 🧻 Logging	Default value (for all clients):			
Logging level	\Map	Browse		
Carlos a filegradu dading     Server file locations     Tutorial files     SEF files	relative to: \\EARTH\K12Composer\lic	Reset Value		
Default trading partner linked direct	Override value for this client installation:			
File locations relative to client Server network path	\Map	Browse		
Map definitions Map generate and compile LANSA Integrator (JSM) instance	relative to: \\EARTH\K12Composer\lic			
Browser		·		

Refer to *System Settings* for further information.

**Note:** You must login to the IBM i server in Windows Explorer in order for these mappings to work.

#### Step 1. Create a Transformation Map Registration

- 1. In the Navigator, expand Definitions and select Transformations maps.
- 2. Select like we from the tool bar or right click and select New from the context menu.



- 3. In the *Transformation map* area:
  - a. Enter *ID*: your initials plus *TUTMAP* (iiiTUTMAP).
  - b. Enter *Description*. For example **Fred Smith LIC tutorial Map 1**.
  - c. Select a *Map Type* from the drop down list. *Map Type* is optional and is defined using *Code Maintenance*. *Map Types* provide a way to for you organize your maps into logical groups.
  - d. For *Restartable*, accept the default *Yes*. This specifies that a *Processing Sequence* containing this map is eligible to be restarted if the *Transformation Map* execution fails.
  - e. For *Status*, select *Active* from the drop down list.

IIITUTMAP	- Eave
Fred Smith Tutorial Map 1	B Refresh
Tutorial exercise	
Yes	Run
Active	
	IIITUTMAP Fred Smith Tutorial Map 1 Tutorial exercise

3. Click the *Save* button.

#### **Step 2. Edit the Transformation Map**

This step will create mapping from an xml file to a database table.

1. Make sure that your *Transformation Map* is selected and the *Details* tab is displayed.

Details Data interchange Parameters Run history Where used Notes Audit     ID IIITUTMAP   Description Fred Smith Tutorial Map 1   Map Type Tutorial exercise   Restartable Yes   Status Active   Click Edit to edit your transformation map.     Click Edit to edit your transformation map, click   Prepare	🏠 Transformation map : De	etails (IIITUTMAP-Fred Smith Tutorial Map 1)
ID       IIITUTMAP         Description       Fred Smith Tutorial Map 1         Map Type       Tutorial exercise         Restartable       Yes<▼	🦳 Details 🛛 🍭 Data intercha	ange   😤 Parameters   <u>]</u> Run history   🔍 Where used   🛄 Notes   🐼 Audit
Description       Fred Smith Tutorial Map 1         Map Type       Tutorial exercise         Restartable       Yes         Status       Active         Click Edit to edit your transformation map.       Edit         Edit       When you have saved your transformation map, click         Prepare       When you have saved your transformation map, click	ID	IIITUTMAP Save um
Map Type       Tutorial exercise         Restartable       Yes         Status       Active         Click Edit to edit your transformation map.       Edit         Edit       Edit         When you have saved your transformation map, click Prepare to complete preparing it for use.       Prepare	Description	Fred Smith Tutorial Map 1
Restartable       Yes         Status       Active         Click Edit to edit your transformation map.       Edit         Edit       Edit         When you have saved your transformation map, click       Prepare         Prepare to complete preparing it for use.       Prepare	Мар Туре	Tutorial exercise
Status     Active       Click Edit to edit your transformation map.     Edit       Edit     Edit       When you have saved your transformation map, click     Prepare	Restartable	Yes Run
Click Edit to edit your transformation map.	Status	Active
When you have saved your transformation map, click Prepare to complete preparing it for use.	Click Edit t	to edit your transformation map.
	When you Prepare to	have saved your transformation map, click Prepare Prepare Prepare

2. Click the *Edit* button on the *Details* tab.

A Warning window may be displayed. This explains that in Altova MapForce you are able to create some mappings that use features that are not supported with LANSA Composer. The mapping you are going to create is from an xml file to a database table. These are fully supported by LANSA Composer. Click the *Continue* button to proceed.

**Note:** After several uses, the Warning window may be hidden by selecting the checkbox at the bottom. If you do not see the warning window it is because you or someone else using this computer has previously hidden it.
Altowa ManEnron FILLT	UTMAD med		
Allova waprorce - [JJJ1	OTMRP.mraj		
Elle Edit Insert Proje	ct <u>C</u> omponent	Connection Function Output View Iools Window Help	- 8 ×
i 🗋 🚅 🖬 💋 🎒 🗠 c	×   X 🗈 🖻	1 🗙 🚧 🤲 😪 🔤 850 850 800 800 00 00 00 00 00 00 00 00 00 00	
: alt fit mit 🗐t 🐎 🛛 ". 🛱 🛆	ter +0 ⇒ 22 (		
Libraries	- 4 ×		~
📰 core	^		
<u>∎</u> -aggregates			
avg	result =		
count	result =		
max	result =		
min	result =		
string-join	result =		
sum	result =		
conversion functions			
boolean	result =		
number	result =		
string	result =		
Classical functions			
a logical functions	1		
equal	result =		
equal-or-greater	result =		
equali-or-less	result =		
greater	result -		
less	result =		
logical pat	result -		
logical-riot	result =		
nogical-or	result -		
inst-organi	result =		~
			>
add	result =	Mapping Database Query Output	
ceiling	result =	INTUTMAP.mfd	4 0 2
divide	result =		
floor	result =	Overview	• 4 >
modulus	result = 🧔		
	· · · · · · · · · · · · · · · · · · ·		
Add Libraries			
ELibraries Project			
MapEorce v2008, Registered to 3:	he tueru (LANS)	(2) (2) (202 2) (2) (2) (2) (2) (2) (2) (2) (2) (2)	NEIM SCOL
hapi orce v2000 Registered to Ju	TELTAOLA (LANDA	V ST2202 MOTO GIUTI	

#### Step 3. Identify the xml source for the Transformation Map

 Insert an XML/Schema File component using either the Insert XML Schema/File toolbar icon or by selecting XML Schema/File from the Insert menu. Navigate to the Composer tutorials directory and select the **TUTorder.dtd** When asked if you wish to supply a sample xml file, reply **Browse** and select TUTorder.xml

If you are not familiar with XML technology you can easily learn more from the Internet. For example http://www.w3schools.com/dtd/default.asp will inform you that:

"The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes."

*TUTorder.xml* is the model source file for your Transformation. The target is going to be database tables.

🙄 Altova MapForce - [JJJTL	JTMAP.mfd*		
Eile Edit Insert Project	<u>Component</u>	C <u>onnection Fun</u> ction Output <u>View T</u> ools <u>Window H</u> elp	- 8×
	V Pa	× 46.45 1955 ; sur sur su have c. c.	
	00 HE H		
່ 🗄າ 🗊າ 📖 🖓 🖓 🕂 🔶 I	🗟 🝈 🗟 🕻	) 🖹 🐏 😳 🔚 🗗 🔁 📅 🍋 🗊 🖬 🚽 100% 💽 🕄 🔀 🛅 🏗 🔟 🕎 🖕	
Libraries	- 4 ×		~
and the second s	^		
		E TUTorder	
avg	result =	¢ ∧ ⊟ () Orders ¢	
count	result =	k - () Sales0rder k	
max	result =	to Solumber to	
min	result =	tþ ⊕ ()Customer tþ	
string-join	result =	CustNumber	
sum	result =	CustName D	
Conversion functions boolean  Lumber  string  equal equal-or-greater equal-or-greater less logical-and logical-or logical-or ond-equal	result = result = result = result = result = result = result = result = result = result =	Contex Conte	
			8
add add	result =	Mapping Database Query Output	
ceiling	result =	Note: The second	4 Þ ×
divide	result =	Overview	
floor	result =		
modulus	result = 🗸		
Add Libraries			
maphonce v2006 Registered to Joh	IT TAOLÀ (L'ANZA	0 @1320-2007 Mitova alibri	NOM SCRL

#### **Step 4. Identify the Target Database for the Transformation Map**

1. Insert a *Database* component using either the *Insert Database* toolbar icon or by selecting *Database* from the *Insert* menu. To connect to the database you must identify or set up an ODBC connection.

**Note:** Altova MapForce runs on your PC and so, usually, the ODBC connection described here needs to connect from your PC to the server on which your application database resides.

Altova MapForce displays a wizard and other panels that allow you to establish the ODBC connection suitable for your server. You need to complete these panels according to the database and drivers you are using. Connecting to databases in Transformation Maps in the Composer User Guide contains information and examples for IBM i and Microsoft Access databases used in these tutorials:

- For the **IBM i server**, the database tables are supplied and installed as \*FILE objects in your partition data library, for example, LICLICLIB. Refer to *Example Database Connections for IBM DB2 for i5/OS* for examples of completing the ODBC connection to the tutorial database on your IBM i Server.
- For the **Windows server**, the database tables are supplied and installed in a Microsoft Access database, TUTORIAL.mdb, in the partition execute directory:

for example, C:\Program Files\LANSA\X\_WIN95\X\_LANSA\X\_LIC\Execute

- Refer to *Example Database Connections for Microsoft Access* for examples of completing the ODBC connection to the Windows server tutorial database.
- 2. The *Select a Database, Connection Wizard* will start. Select *IBM DB2* and click *Next* to accept the IBM DB2 (ODBC) database.



3. On the next dialog, select *ODBC*, and click the *Next* button. (Mapforce 2012 supports either JDBC or ODBC database connection while defining your map.)

	JDBC vs. ODBC
Wizard	
6	© JDBC
Existing	Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The
ADO	enderen nen en ere erekende ne e geven hier aduntud er pendinterhos.
Connections	ODBC
ODBC	An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for
Connections	more modern database features (such as native XML types).
JOBC	
1000	

4. Then select the 'Create a new Data Source Name' option. Click the *Connect* button.

🙄 Select a Databa	se 🛛 🔀
Connection Wizard	Connecting to IBM DB2 Where can I find IBM DB2 drivers?
Existing	Select an option how you wish to connect to the database and click Next. <ul> <li>Create a new Data Source Name (DSN) with the driver:</li> </ul>
Connections	Use an existing Data Source Name: User DSN System DSN Edit Drivers
ADO Connections	
ODBC Connections	
Global Resources	Skip the configuration step for wizard
	< Back Connect Close

5. Enter a *Data Source Name* of <server>\_<library>, where <server> is the IBM i and <library> is the specific library which will be the default for this connection. See later note for details.

Do not click *OK* at this point.

ieries A	cces	s fo	or V	Vind	ow	s Ol	DBC	Set	iup													?	
General	Ser	ver	Pa	ckage	es	Perf	ormat	nce	La	angua	ige	Cat	alog	Т	ran	slatio	on	D	iagn	ostic	]		
Data s	source	e na	me:		-	_			_		-	_	_	_	-	-	1						
EART	гн_к	12P(	GML	IB													)						
Descri	iption															-	1						
iSerie	s Acc	ess	for \	Vindo	ws	ODB	C da	a so	ourc	е													
Syster	m:																						
EART	гн					~	•		(	Conne	ectio	on Op	otions	S									
			~	~					_		~~	-		~					_				~~

6.Click the *Connection Options* button and enter a default User ID:

Connection options
Default user ID Ulse Windows user name Ulse the user ID specified below JIVDRYL35
O Use IBM i Navigator default
O Use Kerberos principal
Signon dialog prompting  Prompt for SQLConnect if needed  Never prompt for SQLConnect
Security
O Do not use Secured Sockets Layer (SSL)
O Use Secured Sockets Layer (SSL)
<ul> <li>Use same security as IBM i Navigator connection</li> </ul>
OK Cancel Help

Click *OK* to close the dialog.

7. Select the *Server* tab. Enter an *SQL default library* of <library> where <library> is the default IBM i library to be used to access the required file(s) during the Prepare Map step.

Series Access for Windows ODBC Setup										
General	Server	Packages	Performance	Language	Catalog	Translation	Diagnostic			
Namin	ig conver	ntion:								
SQL	- naming c	onvention (*	SQL)	~						
SQL d	lefault libr	ary:								
K12P	GMLIB									
Library	list:									
	_			_			_			

8. Click *OK* to connect to the database.

9.Enter your IBM i User ID and Password and click OK.

Signon to	iSeries	? 🛛
	Server:	EARTH
	User ID:	JIVORY
	Password:	
		OK Cancel

The *Signon to iSeries* dialog may be shown up to three times. Complete the login in each case.

10. When you have established your ODBC connection, the mapping tool allows you to choose the particular tables you wish to use in your transformation map:



11. Select the arrow symbol next to the first library shown, to display all the IBM i libraries in a dropdown

Select Tables,Views to Insert							
Source							
▼ ▼ ♦							
EARTH_D11 ODBC							
#RPGLIB   ARCHIVE   AULT2F2   AVHSDATA   BPCSV61   BTF   CHM   CHM32E   CHMCDIRECT<							
Preview							

12. Scroll down to find the library containing the files you want to use in your transformation map. The tutorial database tables can be found in the library for the LANSA partition in which LANSA Composer was installed - for example, D11LICLIB.

Source	
⊽ ⊽ ♦	
D D PSLMST	^
D D PSLTIMES	
🗖 🔟 SKLTAB	
🗖 📶 XBIG810	
🖸 🔟 XN01	~
	<u> </u>

13. Check the box next to files TUTORDH and TUTORDL to mark them as the files you wish to use in your Transformation Map..

**Note:** With a single file selected, the *Preview* button will display the file fields and data. In the example shown below, the file is currently empty:

	PDVSV TUTORDH TUTORDL vs ECT Statements		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Preview Reload	Table: TUTD	RDH	01 ORDDAT	0 UPDATE_DENT	
Hide Preview <<					

Click *OK* to close the dialog. A database component representing the selected files is inserted in the workspace for the map.

**Note:** When used to access IBM DB2 for i5/OS database tables, the library from which you select the files is used only for the purpose of *defining* the transformation map. LANSA Composer will remove the library qualifier from the generated Java code when you prepare the transformation map. You must setup your database configuration such that the required files can be found in the desired library at run-time - you will do this in Step 7. Set up the Database Configuration .

D LANSA01	1
P 🔲 🖽 LICLICLIB.TUTORDH	Ę
C BCHNUM	ե
C ORDNUM	ե
🖒 🔤 CUSNUM	Ľ,
C ORDDAT	Ľ,
PDATE_IDENT	Ľ,
🖕 🛱 🖽 LICLICLIB.TUTORDL	Ľ,
P BCHNUM	ե
🗧 📖 ORDNUM	հ
🖕 👘 ORDLIN	ե
🔁 🔤 🖬 LINPAR	ե
🖒 🔤 🗔 LINPRC	ե
C LINQTY	G
LINVAL	ĥ
UPDATE_IDENT	G

#### Step 5. Map the Input xml to the Target Database

You now have an input xml layout and target database tables in the mapping workspace. In this step you need to make known the connections between the input and target and to identify calculations and additional input.

1. Connect the SalesOrder to the TUTORDH file.

This connection specifies that an order header database record will be created for each <SalesOrder> element encountered in the input XML.

To make the connection, click and hold down the left-mouse button on the *SalesOrder* element, drag the mouse to the right, over the connector arrow adjacent to the TUTORDH file. As you drag the mouse, a connector line will be drawn. When the mouse is over the TUTORDH connector, the mouse pointer will change. Release the left mouse button to complete the connection.

2. Similarly, connect the *Line* to the TUTORDL file.

This connection specifies that an order detail database record will be created for each <Line> element encountered in the input XML.

3. Now connect the data items as follows:



- 4. The field Line Value LINVAL in table TUTORDL is to be derived by a calculation.
  - a. Add a multiply function from the *Math functions* shown on the *Libraries* tab.

	0 3 2 0		
uries	- + ×	C. C. C. Contract	1
5JN	result - A	C ChalesOrder	
Corrections functions bodien number string equal contests gooder legals contests gooder legals contests gooder legals contests gooder legals contests gooder legals contests gooder legals contests legals con	result = result =	Original Statement     Occustores     Occustores     Occustores     Occustores     Occustores     Oster     Ost	
edd	rend -		
celling	reaut -		
divide	resut =		Gatel
1007	result -		Antel (BOA)
103.4.0	reout -	Contraction of the local division of the loc	Summer 1
unitaria	result -	50	
noulld a ddaast	FOTLE -	Mapping Database Guery Out	
PARIEL .	Peaks -	BIIITUTMAP.m/d*	
a⊡node-test functions exists	result -	Destroise	- 0 × Messages
and pullets	rent a		
TAN-PERSONAL STREET	Provide States of the second		

- b. Set the inputs (*value 1* and *2*) as the *Price* and *Quantity* from the input order.
- c. Assign the result to the *LINVAL* field.



- 5. The field BCHNUM in the tables is a batch number. This does not exist in the input but will be provided to the Transformation Map at run-time via a parameter value.
  - a. To set up the mapping for this batch number, select *Insert Input* from the *Function* menu.



- b. Set the *Name* to *BatchNumber* and *Type* to *string*.
- c. Give the batch number a default value of iii123. This will enable the *Output* tab in MapForce to show a preview of the transformation.

Edit Inpu	. 🛛
Name:	BatchNumber
Datatype:	string 💌
	Connection required
Preview	Settings
Spec	cify value
Value:	iii123
	OK Cancel

6. Connect the resulting *BatchNumber* to the BCHNUM in tables TUTORDH and TUTORDL.



- 7. You may click on the *Output* tab to view how the data from the sample input would be added to the database tables. Note that, for the purpose of the preview, the field BCHNUM has the default value you gave it above. At execution time its value will come from a parameter value.
- 8. *Save* your map and exit from *MapForce*.

# **Step 6. Prepare your Transformation Map for use within Composer**

This step will generate Java from the information within your map and identify the parameters required by the map.

1. Make sure that your *Transformation Map* is selected and the *Details* tab is displayed.



- 2. Click the *Prepare* button on the toolbar or on the *Details* tab.
- 3. The *Prepare Transformation Map* dialog is displayed, showing the steps to be performed. Click the *Prepare* button to continue:

👫 Prepare T	ransformation Map		
RG	Transformation map	JMITUTMAP	
35	Description	JMI Tutorial LIC Tutorial Map 1	
	<ul> <li>LANSA Composer will complete t transformation map. Click Prepar</li> </ul>	he steps below to prepare the re to proceed.	Prepare
			Cancel
	6 1 1		
Lheck	folders and files		
Preproc	cess transformation map definition		
Genera	te code for transformation map		
Compile	e code for transformation map		
Latalog	jue information for transformation m	iap - to LANCA Company community	
Depioy	transformation map executable file	s to LANSA Composer server	
Llear w	orking files and folders		

You may see the MapForce window open briefly during the Prepare – this is normal and it will close by itself in a few moments.

4.If the Prepare process seems to be waiting as shown:

🔀 Prepare 1	Fransformation Map		×
ag	Transformation map	JMITUTMAP	
35	Description	JMI Tutorial LIC Tutorial Map 1	
	Preparing transformation map	Prepa	are
		Cano	el
✓ Check ✓ Prepro	folders and files cess transformation map definition		
Gener	rate code for transformation i	map	
Compil Catalo Deploy Clear v	e code for transformation map gue information for transformation m v transformation map executable file vorking files and folders	nap es to LANSA Composer server	

Look for a Signon dialog, which may be hidden, to continue:

然 Prepare T	ransformati	on Map			
	7 Transformation map		JMITUTM JMI Tuto	IAP rial LIC Tutorial M	ap 1
-	Description	IBM i signon			? 💌
	Preparing (r	Sigr	non to:	EARTH	
		Use	er ID:	JIVORYD4C	
✓ Check ✓ Preprod	folders and fil cess transforn	Pas	sword:	•••••	
Gener     Compile     Catalog     Deploy     Clear w	ate code fo e code for trai gue informatio transformatio vorking files ar	n map executable file nd folders	es to LANS/	A Composer serve	OK Cancel

When the preparation steps are completed, a *Map compile has completed* message box is displayed. Click *OK*.

Prepare Ti	ransformation Map	JMITLITMAP	X
X	Description	JMI Tutorial LIC Tutorial Map 1	
	Preparing transformation map		<u>P</u> repare
		LANSA MapCompiler	Cel
Check f     Preproc     General     Compil	olders and files ess transformation map definition te code for transformation map e code for transformation m	Map compile has completed	
Catalog Deploy	ue information for transformation m transformation map executable file orking files and folders	s to LANSA Composer server	

The *Catalog* and *Deploy* steps will be completed. The final dialog may be *Closed* or a detailed log displayed, which is helpful if your *Prepare Map* fails for some reason.

😵 Prepare T	ransformation Map		X
RB	Transformation map	JMITUTMAP	
3	Description	JMI Tutorial LIC Tutorial Map 1	
	Transformation map has been pr	repared successfully.	<u>Close</u> Show <u>Log</u>
<ul> <li>Check</li> <li>Preproc</li> <li>Genera</li> <li>Compile</li> <li>Catalog</li> <li>Deploy</li> <li>Clear w</li> </ul>	folders and files cess transformation map definition ate code for transformation map e code for transformation map gue information for transformation n transformation map executable file vorking files and folders	nap es to LANSA Composer server	

You will now see the parameters from your map have been identified and can be seen on the *Parameters* tab when your map is selected.

4. If you select a different map in the instance list and then re-select your new one, you will see that the following information panel has been updated, showing that this map is ready to be used.



The *Check* button verifies the current status of the transformation map definition and executable files.

The button displays a dialogue that determines the file size and a checksum for the current versions of the *Transformation Map* files (in the file system) and compares them against the file sizes and checksums catalogued when the map was last prepared. Any inconsistencies are highlighted and a Restore function supports restoring the catalogued version of the file from the LANSA Composer database.

- 5. A *Catalogued* command tab in the LANSA Composer client application provides the ability to view the catalogued information for a selected transformation map, including:
  - The current *Prepared* status
  - Information about when and who prepared the map, and the LANSA Composer, Java and Altova MapForce versions that were used

• A visual tree representation of the major components used in the mapping, including database table selections.

Transformation map : Catalogu	ued (IIITUTMAP-iii Tutorial LIC003)	
🖗 Attachments 🛛 🚺 Notes 🛛 🕢 A	udt	
Details 🔍 Data interchange	😤 Parameters 🛛 🞇 Catalogued 🛛 🔂 Run history 🛛 🖶 Version history	🔍 Where used
	Transformation map was successfully prepared by user L35PGMLIB. Clic to verify the current status of the files.	k Check Check
Prepared	2012-02-15 11:23:16 by user L35PGMLIB	
Prepared by	LANSA Composer version 3.5.357	
Prepared using	MapForce version 2012	
Descent day becaut	Java 1 E	
Prepared to target	3440 1.5	
Prepared to target Map definition last saved with	MapForce version 2012	
Prepared to carget  Map definition last saved with	MapForce version 2012 defaultmap1	
Prepared to carget  Map definition last saved with  -Mapping  -Input components	MapForce version 2012 defaultmap1 document	
Map definition last saved with  Map definition last saved with  Mapping  Figure Components  R M M M M M Input	MapForce version 2012 defaultmap1 document Batch_No	
	MapForce version 2012 defaultmap1 document Batch_No	
	MapForce version 2012 defaultmap1 document Batch_No structure	
Map definition last saved with  Map definition last saved with  Mapping  Input components  Mapping  Mapping  Couput components  Database  Database  Data source name	MapForce version 2012 defaultmap1 document Batch_No structure EARTH_L35PGMLIB	
Map definition last saved with  Map definition last saved with  Mapping  Input components  Coutput compon	defaultmap1 decument Batch_No structure EARTH_L35PGMLIB before p	
Map definition last saved with  Map definition last saved with  Mapping  Input components  Couput component	MapForce version 2012 defaultmap1 document Batch_No structure EARTH_L35PGMLIB before p IBM iSeries	
Map definition last saved with  Map definition last saved with  Map definition last saved with  Mapping  Input components  Coutput components  Co	MapForce version 2012 defaultmap1 document Batch_No structure EARTH_L3SPGMLIB before p IBM iSeries (* while	
Map definition last saved with  Map definition last saved wit	MapForce version 2012 defaultmap1 document Batch_No structure EARTH_L3SPGMLIB before p IBM iSeries (* while	
Map definition last saved with  Input  Map definition last saved with  Input  Output components  Database  Database surce name  Strip schema names (t  Database type  Database connection  Database selections  Table  Table  Table	MapForce version 2012 defaultmap1 document Batch_No structure EARTH_L3SPGMLIB before p IBM iSeries (* while TUTORDH TUTORDH	

6. The *Prepare* step will have generated a Java jar file and placed it in the \composer\LIC\map folder within the JSM Instance directory (where LIC is LANSA partition).

# Step 7. Set up the Database Configuration

When the *Transformation* is used within a *Processing Sequence*, it will be executed on your server. It will use the generated jar file and this will use JDBC to connect to the database tables you have used within your *Transformation*. To enable this connection to execute you need to specify a database connection string for use by JDBC. This is done by creating a *Database Configuration*.

- 1. In the *Navigator*, expand *Configurations* and select *Database configurations*.
- 2. Select *New* on tool bar or right click and select *New* from the context menu.
  - a. Enter a *Database configuration ID* **iiiTUTDBID** (where iii is your initials).
  - b. Enter a *Description* such as **Fred Smith Tutorial database configuration**.
  - c. Set the Status to Active.
- 3. Enter a *Database connection string*.

A connection string for an IBM i server known as MYSERVER would be:

jdbc:as400://MYSERVER/MYLIB;naming=sql;errors=full;

date format=iso;

translate binary=true

where *MYSERVER* is the name of the IBM i server and *MYLIB* is the name of the library containing the database tables you wish to access.

The following alternate form for the connection string permits database tables from more than one library (or schema) to be used. The JDBC connection property naming=system must be specified for this to be effective:

jdbc:as400://SYSNAME;naming=system;libraries=LIB1, LIB2,\*LIBL;errors=full;date format=iso; translate binary=true The tutorial database tables can be found in the library for the LANSA partition in which LANSA Composer was installed - for example, LICLICLIB.

The connection string for the Microsoft Access database on a Windows server would be similar to this:

jdbc:odbc:;DRIVER=Microsoft Access Driver (\*.mdb);DBQ=C:\Composer\Tutorial\Tutorial.mdb

4. Enter a *Database driver class*.

For an IBM i server this would usually be

com.ibm.as400.access.AS400JDBCDriver

For a Windows server this would usually be

sun.jdbc.odbc.JdbcOdbcDriver

Refer to *Database Configuration* and supplied example database configurations for more information and examples of JDBC connection strings and driver classes for common databases.

5. Enter a *Database user* and *Database password*. This must be a valid user on your server which has authority to the tables you are using within your map. (When using the Microsoft Access database on a Windows server, the database user and password should not be necessary.)

Your systems administrator will be able to provide a User ID and Password.

6. Set *Commit automatically* to *No*. Set *Transaction isolation* to *None* 

Your Database Configuration should now look similar to the following:

🔞 Database configuration : Detai	ils (IIITUTDBID-iii System i database configuration)	
Details 🖉 Attachments 门 N	otes 🕢 🕢 Audit	
ID	IIITUTDBID	Save
Description	iii System i database configuration	Befresh
Status	Active	- Norrosh
Database connection string	ql;errors=full;date format=iso;translate binary=true	
Database driver class	com.ibm.as400.access.AS400JDBCDriver	
Database user	L35PGMLIB	
Database password	*****	
Commit automatically	No	
Transaction isolation	None	
		~~

The necessary settings for these fields depends on options used in the *Transformation Map* definition and in the JDBC connection string and on the JDBC driver implementation but these are usually the correct values when accessing the database tables on IBM i using the IBM-supplied JDBC driver.

7. *Save* your database configuration.

#### Summary

- You have created a *Transformation Map* using *Altova MapForce*. If you haven't already done so, it is worthwhile working through the tutorials provided with MapForce to familiarize yourself with the features available in the mapping.
- You have prepared your Transformation map so it is available for use within Composer.
- You have identified how to connect to the database tables used in your Transformation Map with a Database Configuration. This identifies the JDBC connection that will be used by Composer when your Transformation Map is executed on your server.
- **Reminder:** The connect to the Database tables within *MapForce* uses an ODBC connect and connects from your PC client to your server.

The connect to the Database tables when the *Transformation Map* is executed within Composer uses a JDBC connect and will execute on the server.

- LANSA Composer does not provide the database connectivity components (ODBC and JDBC drivers). You must ensure that you have the appropriate drivers for the databases you wish to use with Composer and documentation for the drivers, including how to setup the connection strings. Your database software vendor will normally provide these. For DB2 on an IBM i server you should refer to the IBM documentation for System i, DB2 and their JDBC driver.
- Refer to *Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS* in the Composer online guide for more information. The database connections string provided in this example will not require that your physical files are journalled. If you require the file changes made via the transformation map to be journalled, you should refer to the information in the guide.

# LIC004 - Add the Transformation to the Processing Sequence

## Objective

• To add your *Transformation Map* to the *Processing Sequence*. The *Transformation Map* will be applied and the data added to the database tables for each xml file that has been found in the tutorial directory.

# **Before You Begin**

In order to complete this tutorial, you should have completed the following

- LIC001 Introduce Composer Client & Validate Environment
- LIC002 Create a Processing Sequence
- LIC003 Create a Transformation Map

# Steps

To achieve these objectives, you will complete the following:

- Step 1. Set up a Loop to Cycle through the List of xml Files
- Step 2. Add the Transformation to the Processing Sequence
- Step 3. Execute your Amended Processing Sequence
- Step 4. Add a Variable Batch Number to the Processing Sequence
- Summary

#### Step 1. Set up a Loop to Cycle through the List of xml Files

- 1. Navigate to *Definitions/Processing Sequences* and select the *Processing Sequence* you created in tutorial LIC0002 *iiiTUTSEQ01*.
- 2. Press the *Copy* button to create a new processing sequence based on this one:



3. Give the new processing sequence an *ID* of iiiTUTSEQ02 and click the *Copy* button.



When the details for your copied iiiTUTSEQ02 are shown, give the new processing sequence an appropriate description as shown:

<table-of-contents> Processing sequence : D</table-of-contents>	etails (JMITUTSEQ02-JMI Tutorial - LIC002)	
O Audit Details Run commands	🔁 Run history   🖶 Version history   🍢 Cross references   🖉 Atta	achments 🚺 Notes
Name	JMITUTSEQ02	Save
Description	JMI Tutorial - LIC004	Refresh
Restartable	Yes	Run
Status	Active	
Event identifier		

- 4. Save the change to your new processing sequence.
- 5. Click the *Edit* button to open the Processing sequence editor window. Delete the activity TUT\_02\_AT from the previous exercise, so that your sequence contains a single Activity DIRECTORY\_LIST. The *Activity* will output a list of files in a list named FILELIST.
- 6. Select the *Parameters* tab and delete the YOURINITIALS input parameter.
- 7. From the *Palette* tab select the *Loop* construct and add it to the sequence *at the end*. Your sequence will now contain the Activity and then Loop. Note that the *Error* tab (bottom left) shows errors (red triangle) and warnings (yellow triangle). This is because the details for the Loop have yet to be entered.

- 8. Select the *Loop*. The details of the loop will be shown (bottom right).
  - a. In the *List name* enter *FILELIST*.
  - b. In the *Loop variable* enter *THISFILE*. This means that the loop will cycle through the list of entries in the FILELIST and the value of the entry which is currently being processed will be set in the variable named THISFILE. The variable THISFILE may be used as an input to subsequent processing steps within this loop.
  - c. In the *Index variable* enter FILECNT. The variable FILECNT will contain the number of the entry which is currently being processed.

Note that the previously displayed errors (red triangle) have gone, though a warning (yellow triangle) may remain about the loop item being empty - you

will deal with this next.



#### Step 2. Add the Transformation to the Processing Sequence

The Transformation is to be applied to each of the xml files in the *FILELIST*.

1. Select the *Transformations tab* (top left). In the list of Transformations shown, select your tutorial map (iiiTUTMAP) and add it to the Processing Sequence *after the Loop*. Your Transformation should be nested within the Loop construct. If you select the \*\*\* End of processing sequence \*\*\* text before dropping the transformation map into the processing sequence editor, you'll ensure the Loop is placed at the end.

**Note:** You may use the Promote  $\triangleleft$  Promote or Demote  $\triangleleft$  buttons on the tool bar to adjust the level of nesting.

- 2. Select the *Transform* in the sequence details. The parameters for this Transformation will then be shown in the *Parameters* tab (bottom right). Against these parameters you need to identify which values and variables are to be used.
- 3. For the *SourceFileName*, set the *Variable or value for parameter* to **&THISFILE**. You may select a variable from the *Variables* tab (top left) and drag and drop it onto the Variable or value for parameter.
- 4. For the *BatchNumberSourceParameter* set the *Variable* or *value for parameter* to iii**123**. For the input Batch Number for this tutorial, use a constant value of iii**123**.
- 5. For the *DBTargetConnection* set the *Variable or value for parameter* to the name of the DataBase Connection Id you created in LIC003, Step 7. Set up the Database Configuration (iiiTUTDBID). You may also select the configuration from the *Configurations* tab (top left) and drag and drop the required configuration onto the *Variable* or value for parameter.

literr	n		Details		-
			*** Beginning of processing segue	ence ***	
		Activity			
	1	Vame:	DIRECTORY_LIST - List files in a c	directory	
8 (	ð i	Loop			
	L	.ist:	8FILELIST		
	l	.oop variable	&THISFILE		
	1	ndex variable	a &FILECNT		
		🔓 Transfo	m		
		Name:	IIITUTMAP - ii Tutorial LIC003		
			*** End of processing sequence *	**	
	-				
Det	ails	Parameters			
			Variable or value for parameter	Parameter Name	Parameter description
	*	Inbound	&THISFILE	IIITUTMAP_1	TUTorder2SourceFilename
1	_	Table and	INUS22	TITLITMAD 2	Batch MoSourceBarameter
1	+	Incound	JMI123	IIIIOIPPP_2	Datch_N000urcerarameter

6. *Save* your Processing Sequence.

## Step 3. Execute your Amended Processing Sequence

- 1. Before clicking on the *Run* icon, make a note of the path for the Composer Tutorial files. You can do this by accessing System Settings and using Ctrl-C to copy the value you need.
- 2. To execute the *Processing Sequence*, click the *Run* button on the toolbar.
- 3. The *Run Processing Sequence* window is shown. Select the *Run immediately* radio button.
- 4. In the *DIRECTORY* parameter, enter the path to the LANSA Composer Tutorial as used in previous exercises. Note there is no / at the end of this path.

5	Processing sequence	JMITUTSEQ02
	Description	JMI Tutorial - LIC004
	O Submit to <u>b</u> atch	Run immediately
	Type parameter values	s for this run:
	Directory to be listed	/lansa_composer_k12pgmlib/lic/tutoria
Close	📄 📄 R <u>u</u> n	Cancel <u>S</u> ave shortcut

5. Click on the *Run* button at the bottom of this window.

When the *Processing Sequence* completes, the steps and results of the processing are shown in the *Processing Sequence Log* window. Verify that your processing sequence ended successfully by reviewing the log.

6. Check the database tables, TUTORDH and TUTORDL. The data from the xml input file will have been added to the tables. The Batch number is iii*123* (from the constant value you assigned to the batch number) in Step 2. Add the Transformation to the Processing Sequence.

The following screen pictures show an SQL query on files TUTORDH and TUTORDL.

For example: SELECT \* FROM LICLICLIB/TUTORDH

#### Order Header TUTORDH:

- EARTH - newloo	k					
e Session Edit View Macros Icols Function Keyboard Window Help						
1 a a 1 a a	' 🖉 🖶 🗼 🕩 🛍 🖉 🔳 🖬 😰 🕵 🕼 🕼 🕼 🕼 🕼 🤣					
Display Data						
Data width						
Position to line	Position to line Shift to column					
+1.	+2.	+	++	5+		
Batch	Order	Customer	order	Update/		
Number	Number	Number	date	Identifier		
0000000013	12345	543	061005	Θ		
0000000015	12345	543	061005	Θ		
123	12345	543	061005	Θ		
123	12345	543	061005	θ		
123	12345	543	061005	Θ		
123	12345	543	061005	Θ		
123	12345	543	061005	Θ		
123	12345	543	061005		-	

#### Order Lines TUTORDL:

1 - EARTH - newlook					
Ele Session Edit Yew Macros Icols Function Keyboard Window Help					
//#INE/#INE/#INE/#####					
Display Data					
				Data width	
Position to line Sh			Shift to column		
+ 1	+ 2	+ 3	+ 4 +	5 + 6	+ 7 +
Batch	Order	Order	Part	Line	Line
Number	Number	Line	number	Price	Quantity
		Number			
000000013	12345	1	123	9.95	10
000000013	12345	2	456	13.27	5
000000015	12345	1	123	9.95	10
000000015	12345	2	456	13.27	5
123	12345	1	123	9.95	10
123	12345	2	456	13.27	5
123	12345	1	123	9.95	10
123	12345	2	456	13.27	5

# **Step 4. Add a Variable Batch Number to the Processing Sequence**

In this step, instead of the constant value of iii**123** for the Batch number, you will change the *Processing Sequence* to use a next number activity. This way you can tell which is the latest addition to the tables TUTORDH and TUTORDL.

- 1. On the *Activities* tab, locate the **NEXTNUMBER** *Generate the next number* activity, in the *Variable Manipulation* group. .
- 2. Add (double-click or drag and drop) this *Activity* to the sequence before the *Transform*. We will use this *Activity* to assign a variable BATCHNO.
- 3. Select the *NEXTNUMBER Activity* in the sequence. The parameters for this Activity will then be shown in the *Parameters* tab (bottom right). Against these parameters you need to identify which values and variables are to be used.
  - a. For the *NUMBERKEY1* parameter, set the *Variable or value for parameter* to '**TUTORIAL**'. You may type the value with or without the quote marks. If you do not type the quote marks, LANSA Composer will add them later.
  - b. For the *NUMBERKEY2* parameter, set the *Variable or value for parameter* to **'BATCHNO'**. Using these two "key" values means that the batch number will be generated from a series that is common to all processes that use the same values (that is, a series that is common to all users of these tutorial exercises).
  - c. For the *NUMBEROUT* parameter, set the *Variable or value for parameter* to **&BATCHNO**. This means that the number assigned by this activity will be placed in a processing sequence variable named BATCHNO.
- 4. Select the *Transform* in the sequence.
  - a. In the *Parameters tab* (bottom right), find the parameter *iiTUTMAP\_2 BatchNumberSourceParameter*.
  - b. Delete the Parameter value of iii**123**. For the *Variable or value for parameter* enter **&BATCHNO** or drag and drop from the Variables tab (top left).

This means that the value for *BATCHNO* output by the Activity *NEXTNUMBER* will be used as the input value by the Transformation.

-	*** Beginning of processing sequence ***							
	÷	Activity	vity					
Name: DIRECTORY_LIST - List files in a Directory				Ę				
	🛛 🍈 Loop							
		List: &FILELIST						
	Loop variable: &THISFILE				)			
		Index variable	e: &FILECNT		1			
		🍰 Activity			1			
		Name:	NEXTNUMBER - Generate the next number	NEXTNUMBER - Generate the next number				
		ᡖ Transfo	rm		1			
Name:			JMITUTMAP - Tutorial - LIC003	JMITUTMAP - Tutorial - LIC003				
*** End of processing sequence ***					\$			
_								
Del	tails	Parameters			Ĵ			
			Variable or value for parameter Parameter Name Para					
1	*	Inbound	&THISFILE JMITUTMAP_1 TUTor					
2	*	Inbound	&BATCHNO JMITUTMAP_2 Batch					
3	*	Inbound	JMIDB_400' JMITUTMAP_3 EARTH					
	<b>.</b>			~~~~~				

- 5. Save your Processing Sequence.
- 6. *Run* your *Processing Sequence*. Look at the log (*View Log* tab) to check the batch number allocated:



- 7. Check the database tables, TUTORDH and TUTORDL again, using SQL. You will see that additional entries have been added to the tables and that the Batch Number is a different number.
- 8. *Exit* the Processing sequence editor.

# Summary

- You have used more of the facilities available within a Processing Sequence:
  - The flow control construct of LOOP to repeat a sub set of processing steps.
  - The technique of the output from one processing step being input to another processing step.

# LIC005 - Set up a Trading Partner

## Objectives

- To introduce you to *Trading Partner* information.
- To link directories to the *Trading Partner*
- To extend your *Processing Sequence* to use the *Trading Partner* structure.

# **Before You Begin**

In order to complete this tutorial, you should have completed the following:

- LIC001 Introduce Composer Client & Validate Environment
- LIC002 Create a Processing Sequence
- LIC003 Create a Transformation Map
- LIC004 Add the Transformation to the Processing Sequence

#### Steps

To achieve these objectives, you will complete the following:

- Step 1. Create a Trading Partner
- Step 2. Set up the Directories that your Trading Partner will use
- Step 3. Add the Trading Partner to your Processing Sequence
- Step 4. Execute your Processing Sequence
- Summary

## **Step 1. Create a Trading Partner**

- 1. In the *Navigator* expand *Definitions* and select *Trading partners*.
- 2. Select *New* on the tool bar or right click and select *New* from the context menu.

🍇 Trading partner		
ID	JMITUTTP	💾 Save
Description	Fred Smith Tutorial Trading Partner	Refresh
Status	Active	
Combod.	End on the	
Contact	Fred Smith	
Address 1	123 AnyStreet	
Address 2	Anytown	
Address 3		
Email	me@mycompany.com	
Phone number		
Fax number		

- 3. Enter *Trading Partner ID* as **iiiTUTTP** (where iii are your initials).
- 4. Enter *Trading Partner Name*, for example **Fred Smith Tutorial Trading Partner.**
- 5. Set the *Status* to *Active*.
- 6. For *Email* enter your own email address. As you are going to send an email to this address, it needs to be an email address that you can access.
- 7. You may enter additional details such as Contact, Address, Phone number, Fax.
- 8. Press the *Save* button.
- 9. Composer will now optionally create a set of folders, suitable for storing documents which are received, processed and sent to the partner.

🔹 Trading Partner IIITUTTP - Add Linked Directories						
LANSA Composer can link directories of the types shown below for this trading partner. If necessary, LANSA Composer can also create the						
spec	ciriea	directories.				
Che	Check the directory types that you want LANSA Composer to link to this trading partner. Type the directory path to use for each selected					
	, ory (	ypo.				
			Directory Type	Directory Path		
1	~	IDOC_ARCHIVE	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_ARCHIVE/		
2	<ul> <li>Image: A start of the start of</li></ul>	IDOC_INERROR	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_INERROR/		
3	<b>&gt;</b>	IDOC_PROCESS	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_PROCESS/		
4	<ul> <li>Image: A start of the start of</li></ul>	IDOC_RECEIVE	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_RECEIVE/		
5	<b>v</b>	ODOC_ARCHIVE	Outbound transaction d	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/0D0C_ARCHIVE/		
6	<b>~</b>	ODOC_INERROR	Outbound transaction d	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/0D0C_INERROR/		
7	<b>~</b>	ODOC_PREPARE	Outbound transaction d	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/ODOC_PREPARE/		
8	8 ODOC_SEND Outbound transaction di /LANSA_Composer_k12pgmlib/lic/IIITUTTP/0D0C_SEND/					
Create directories if they do not exist						

10.Click *OK* to create these folders.

These steps have set up the basic information about your *Trading Partner*.

## Step 2. Set up the Directories that your Trading Partner will use

- 1. Select your tutorial *Trading Partner*.
- 2. Select the *Linked Directories* tab. Note that the IDOC\_ and ODOC\_ set of directories have been create as Linked Directories.
- 3. Create a new linked directory for this *Trading Partner* by clicking the *Add* button on the right of the *Linked Directories* tab to open a separate dialog containing a list of Directory types. This list of directory types is built from the *Code Values* (to view this in the *Navigator* expand *Administration and Housekeeping* and select *Code Maintenance*, then *Code group* of Trading Partner directory types).
- 4. From the list of *Directory Types* select *Tutorial*. To select, use the checkbox then click *OK*.

🕴 Trading Partner IIITUTTP - Add Linked Directories					
LANSA Composer can link directories of the types shown below for this trading partner. If necessary, LANSA Composer can also create the specified directories.					
Check the directory types that you want LANSA Composer to link to this trading partner. Type the directory path to use for each selected directory type.					
	Directory Type Directory Path				
1 🗹	TUT Tutorial /LANSA_Composer_k12pgmlib/lic/IIITUTTP/TUT/				
Create directories if they do not exist OK Cancel					

The *Directory Types* window will close and your selection will be placed in the list of linked directories.

5. The *Directory Path* will be set to a path based on partner name and the code for the *Directory Type*, relative to the *Default Trading Partner Linked Directory* in system settings.

For example:

```
/LANSA_Composer_licpgmlib/lic/IIITUTTP/TUT
```

Where iiiTUTTP is the *Trading Partner* name.

- 6. Press the *OK* button to save your changes.
- 7. Copy the order file TUTORDER.XML from the tutorial folder. For example, /LANSA\_composer\_licpgmlib/lic/tutorial, into your trading partner's
# linked tutorial folder /LANSA\_composer\_licpgmlib/lic/IIITUTTP/TUT

12	📅 Outbound numbering 🛛 🔯 Notes 🕢 Audit						
	Details 🛛 🎯 Group	ps 🛛 🕥 Properties 🛛 👒	Data interchange 😫 Linked directories 🖓 Linked maps 🕼 Linked configural				
		Directory Type	Directory Path				
1	IDOC_ARCHIVE	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_ARCHIVE/				
2	IDOC_INERROR	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_INERROR/				
3	IDOC_PROCESS	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_PROCESS/				
4	IDOC_RECEIVE	Inbound transaction doc	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/IDOC_RECEIVE/				
5	ODOC_ARCHIVE	Outbound transaction do	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/ODOC_ARCHIVE/				
6	ODOC_INERROR	Outbound transaction do	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/ODOC_INERROR/				
7	ODOC_PREPARE	Outbound transaction do	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/ODOC_PREPARE/				
8	ODOC_SEND	Outbound transaction do	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/ODOC_SEND/				
9	TUT	Tutorial	/LANSA_Composer_k12pgmlib/lic/IIITUTTP/TUT/				
	i						

#### **Step 3. Add the Trading Partner to your Processing Sequence**

In this step you will define a processing sequence to do the following:

- Loop through all trading partners
  - a. If a trading partner tutorial directory is defined (is not equal to blank)
  - b. If trading partner name = iiiTUTTP
  - i. List XML files in trading partner tutorial directory
  - ii. Loop through list of XML files
  - iii. Assign a batch number

iv. Transform XML file to database files order header (TUTORDH) and order details (TUTORDL)

- 1. In the *Navigator*, expand *Definitions* and select *Processing sequences*. Select your *Processing Sequence* **iiiTUTSEQ02**. As before, create a new processing sequence by copying this one. Save it with an ID: of iiiTUTSEQ03 and Description: iii Tutorial LIC005
- 2. Click the *Edit* button to open the *Processing Sequence* in the editor.
- 3. From the *Palette* tab select the *Loop* construct and add it to the sequence **at the beginning**.
- 4. Select the new loop and enter the *Details* (bottom right). Enter **List name \*TRADINGPARTNERS Note:** This must end with an "S".

Alternatively, select the *Built-ins* tab (top left) and drag and drop **\*TRADINGPARTNERS** to the List name.

- 5. Demote, using the toolbar icons, the other entries in the sequence to be below the *Trading Partners* loop. This is telling the processing, to loop through the *Trading Partners* and execute the child entries for each *Trading Partner*.
- 6. From the *Palette* tab, select the *If* construct and add it to the sequence *after* the *Trading Partner* loop and *before* the *Activity* **DIRECTORY\_LIST**. Demote the entries after the inserted **If**.
- 7. Filter the loop to only review *Trading Partners* with your partner ID. Select the *If* you just added and in the *Details* (bottom right) enter the *Condition:*

\*TRADINGPARTNER.ID = 'iiiTUTTP' (where iii are your initials)

Once again, note that "tradingpartner.id" can be dragged from the *Built-ins* tab. It is found under *Trading partner attributes*.

Your Processing S	Sequence s	hould now	7 look like tl	he following:

Item	Details
	*** Beginning of processing sequence ***
🗆 🍈 Loop	
List:	*TRADINGPARTNERS
⊟ <del>[}</del> If	
Condition:	*TRADINGPARTNER.ID='JMITUTTP'
🖃 🌧 Activity	
Name:	DIRECTORY_LIST - List files in a Directory
🗆 🍊 Loop	1
List	&FILELIST
Loop variable:	&THISFILE
Index variable:	&FILECNT
🖃 🍰 Activity	
Name:	NEXTNUMBER - Generate the next number
🖃 🖾 Transform	
Name:	JMITUTMAP - Tutorial - LIC003
	*** End of processing sequence ***
Share a state of the second second second	الأسيب والمحمول والمحمول والمحمول والمحمول والمحمل والمحمول

You now need to adjust the *Processing Sequence* to execute the child entries only if the *Trading Partner* has a linked tutorial directory.

- 8. From the *Palette* tab, select the *If* construct and add it to the sequence immediately *after* the *Trading Partner* loop and *before* the If construct added in 6 above. Demote the entries after the inserted *If* you have just inserted.
- 9. Select the *If* you just added and in the *Details* (bottom right) enter the *Condition:*

\*TRADINGPARTNER.DIR.TUT \*NE ''

**Note:** The *\*TRADINGPARTNER* does not end with "*S*". This variable refers to the current *Trading Partner* which is the current entry of the *\**TRADINGPARTNERS loop.

10.Select the *Activity* **DIRECTORY\_LIST**. For the *Parameter DIRECTORY* (bottom right) for the *Variable or value for parameter* enter the value **\*TRADINGPARTNER.DIR.TUT** 

For more information on this entry refer to *Processing sequence* then the *Built-in Variable* topic in the Composer Guide.

11.You are now using the directory linked to the Trading Partner as the DIRECTORY in the DIRECTORY\_LIST. Previously this was an input input

parameter to the processing sequence. You can now remove this by selecting *Parameters* tab (top right) and deleting the *DIRECTORY* and YOURINITIALS entries. The initial *Parameters* tab (top right) should now contain no entries.



Your Processing Sequence should now look like this:

12.Save your Processing Sequence.

#### **Step 4. Execute your Processing Sequence**

- 1. To execute the *Processing Sequence*, click the *Run* button on the toolbar.
- 2. The *Run Processing Sequence* window is shown. Select the *Run immediately* radio button. Notice that you are no longer requested to enter a value for DIRECTORY.

🕨 Run Proc	essing Sequence		
1	Processing sequence Description	JMITUTSEQ03 JMI Tutorial - LIC005	
	O Submit to <u>b</u> atch	💿 Run immediately	
	Bun	ancel	Close
amaaaaaaaaaaaa			

3. Click on the *Run* button at the bottom of this window.

When the *Processing Sequence* completes, the steps and results of the processing are shown in the *Processing Sequence Log* window.

- a. Verify that your processing sequence ended successfully by reviewing the log.
- b. Click the *Show All* button to see full details of all the processing that was performed including the loop instances that processed other Trading Partner definitions.
- c. Check the contents of the database tables TUTORDH and TUTORDL. You should see additional entries have been added.

*LANSA Composer* provides another way of reviewing the results of your *Processing Sequences*. The *Operations Console* can be used by anybody with a web-browser and the necessary network access.

a. From the Navigator, select Operations / Console.

Console						
Refresh	Open i	n Web Browser				
Com	LAN IPO	ser				
Processing Seque	nces	Run History				
Run Histor	y	↓ Later	Earlier )			
Name		Run Number	Start Date / Time	End Date/Time	Sta	itus
			2009-01-14 💷		All	•
JMITUTSEQ03	Q	00000000001211	2009-01-14 13:18:05	2009-01-14 13:18:10	0	ок
JMITUTSEQ02	Q	00000000001210	2009-01-14 11:54:51	2009-01-14 11:54:53	0	ок
JMITUTSEQ02	Q	00000000001209	2009-01-14 11:52:50	2009-01-14 11:52:53	0	ER
JMITUTSEQ02	Q	00000000001208	2009-01-14 11:40:45	2009-01-14 11:45:34	0	ок
JMITUTSEQ01	Q	00000000001207	2009-01-13 15:52:26	2009-01-13 15:52:29	0	ок

- b. Open the required processing sequence by selecting the *Run Number*:
- c. Click the *Show All* button to display the full processing sequence log information:

Processing sequence       JMITUTSEQ03         Description       JMI Tutorial - LIC005         Run number       0000000001211         Start date / time       2009-01-14 13:18:05         End date / time       2009-01-14 13:18:10         Execution result code       OK         Condition for If item tested TRUE       Message ID DXS0004         Beginning If block       Condition for If item tested FALSE         Execution LOOP instance 6       *COMP         Wessage 1 files listed from tested TRUE       Directory /LANSA_Composer/Tutorial to Beginning If block         Condition for If item tested TRUE       Directory /LANSA_Composer/Tutorial to Beginning If block         Beginning If block       Directory /LANSA_Composer/Tutorial to Beginning If block         Beginning If block       Beginning If block         Beginning If block       Directory /LANSA_Composer/Tutorial to Beginning If block         Beginning If block       Beginning If block         Beginning If block       Beginning If block         Beginning If block       Directory /LANSA_Composer/Tutorial to Beginning If block         Beginning If block       Beginning If block         Beginning If block       Beginning If block
Description       JMI Tutorial - LIC005         Run number       000000001211         Start date / time       2009-01-14 13:18:05         End date / time       2009-01-14 13:18:10         Execution result code       OK         Execution result code       OK         Image: Condition for If item tested TRUE       Image: Condition for If item tested FALSE         Image: Executing LOOP instance 6       * COMP         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE         Image: Condition for If item tested TRUE       Image: Compensation for If item tested TRUE
Run number       000000001211         Start date / time       2009-01-14 13:18:05         End date / time       2009-01-14 13:18:10         Execution result code       OK         Condition for If item tested TRUE       Message ID DXS0004         Execution LOOP instance 6       * COMP         Execution for If item tested TRUE       Message 1 files listed from tested from tested TRUE         Executing LOOP instance 6       * COMP         Condition for If item tested TRUE       Message 1 files listed from tested from tested TRUE         Beginning If block       Directory /LANSA_Composer/Tutorial Log date         Beginning If block       Condition for If item tested TRUE         Beginning If block       Directory /LANSA_Composer/Tutorial Log date         Beginning If block       Condition for If item tested TRUE         Beginning If block       Starting activity DIRECTORY LIST         Activity ID       DIRECTORY_LIST
Start date / time       2009-01-14 13:18:05         End date / time       2009-01-14 13:18:10         Execution result code       OK         Execution for If item tested TRUE       Message ID DXS0004         Executing LOOP instance 6       * COMP         Execution for If item tested TRUE       Message 1 files listed from tested TRUE         Executing LOOP instance 6       * COMP         Message 1 files listed from tested TRUE       Directory /LANSA_Composer/Tutorial Log date 2009-01-14 13:18:07         Execution for If item tested TRUE       Message 1 files listed from tested TRUE         Beginning If block       Directory /LANSA_Composer/Tutorial Log date 2009-01-14 13:18:07         Activity ID DIRECTORY_LIST       Activity List files in a Directory
End date / time 2009-01-14 13:18:10 Execution result code OK Execution for If item tested TRUE Condition for If item tested FALSE Executing LOOP instance 6 Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Beginning If bl
Execution result code OK  Condition for If item tested TRUE  Condition for If item tested FALSE  Executing LOOP instance 6  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Condition for If item tested TRUE  Condition for If item tested TRUE  Beginning If block  Condition for If item tested TRUE  Condition for If
Condition for If item tested TRUE Beginning If block Condition for If item tested FALSE Condition for If item tested FALSE Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Condition for If item tested TRUE Beginning If block Condition for If item tested TRUE Condition for If item tested TRU
Beginning If block     Condition for If item tested FALSE     Executing LOOP instance 6     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Beginning If block     Condition for If item tested TRUE     Starting activity DIFECTORY LIST
Activity starting: List files in a Directory     Library list changed     Open connection to the JSM     Load FILECLIENT Service (Integrator command)     LIST Directory (Integrator comand)     JSM *SERVICE trace data is available     JSM service unload     Close connection to the JSM
UIST Directory Summary

#### Summary

- You have used the *Trading Partner* structure which enables you to subset your information directories, Transformation Maps, configurations.
- You have used the *Trading Partner* structure within your *Processing Sequence*. This give you access to the pre-defined variables associated with the *Trading Partners*.
- You are aware that detailed log information for a processing sequence can be accessed via the *Operations / Console*, and that this facility is available to users who do not have the *LANSA Composer* client software installed.

# LIC006 - Add Email Acknowledgement to Processing Sequence

#### Objective

• To introduce you to configurations that can be used within *LANSA Composer* by adding an email notification. The email will be sent to the email address for the Trading Partner.

To complete this tutorial you must be able to email using SMTP from your server.

#### **Before You Begin**

In order to complete this tutorial, you should have completed the following:

- LIC001 Introduce Composer Client & Validate Environment
- LIC002 Create a Processing Sequence
- LIC003 Create a Transformation Map
- LIC004 Add the Transformation to the Processing Sequence
- LIC005 Set up a Trading Partner

#### Steps

To achieve these objectives, you will complete the following:

- Step 1. To set up the SMTP Server Configuration
- Step 2. Set up the SMTP mail detail configuration
- Step 3. Add an Email acknowledgement to your Processing Sequence
- Step 4. Execute your Processing Sequence
- Summary

#### Step 1. To set up the SMTP Server Configuration

The SMTP email within Composer uses the base *LANSA Integrator* SMTP Mail Services.

The SMTP email facility within Composer consists of two parts:

- The first part sets up the SMTP server configuration.
- The second part sets up details of individual email messages.
- 1. To set up the *SMTP server configuration*, expand *Configurations* in the *Navigator* and select *SMTP server configurations*.
- 2. Select *New* on tool bar or right click and select *New* from the context menu to open the Details tab in the SMTP window.
  - a. Enter an *SMTP Mail Server ID*, for example iii**TUTSMTPID**.
  - b. Enter a Description.
  - c. Set the *Status* to *Active*.
  - d. Enter the *SMTP server*. This is the host name of your SMTP server. (For example **10.2.1.99** or **MAILSERVER**).

If necessary consult your systems administrator for the correct email server to use.

The remaining the values are optional and will depend on the setup of the mail server you are using. Enter the values that are required for your environment.

Your SMTP Server configuration should now look like the following:

By ID By Status By Server		SMTP server ID 🔺	Status	SMTP Server
	AB count	DEFAULT_SMTP	Active	123.123.123.123
ClearList	ing search	JMITUTSMTPID	Active	jupiter.lansa.co.uk
SMTP Mail Server ID				
SMTP server configuration			_	
ID	IIITUTSMTPID			E Save
Description	iii Tutorial			Refresh
Status	Active			- nemestr
SMTP Server	jupiter@lansa.co.uk			
SMTP server user profile	mailuser			
SMTP server password	****			
TCP/IP port				
Use secure connection (SSL)	No			
Mail domain				
Character set for body & subject				
Signer				
Encryption algorithm				
Content type				
Encoding				
Trim				
Truncate	V			

3. *Save* your SMTP server configuration.

**Note:** If you have previously used LANSA Integrator SMTP services, you will have already set up the STMP server information within the SMTPMailService properties file. You may use this within LANSA Composer. If the MAIL\_SEND Activity within Composer is used and no SMTP server configuration ID is provided and there is no default SMTP server configuration specified in the System Settings, then the information from the LANSA Integrator SMTPMailService properties file will be used.

#### Step 2. Set up the SMTP mail detail configuration

This step sets up the details of the email message that is to be sent.

- 1. To set up the *SMTP mail detail* configuration, expand *Configurations* in the *Navigator* and select *SMTP mail details*.
- 2. Select *New* on tool bar or right click and select *New* from the context menu.
  - a. Enter the *Email detail ID* as *iiiTUTEMAILACK* where iii are your initials.
  - b. Enter a *Description*. (For example **Fred Smiths Tutorial Email Acknowledgement**.)
  - c. Set the Status to Active.
  - d. Enter *to@mycompany.com* in the *To Email Address*. (This e-mail address will not be used in this tutorial instead the e-mail address in the trading partner definition will be used).
  - e. You could also provide *CC* and a *Blind copy* email address, but for this exercise leave them blank.
- 3. The *From email address* will have already been completed for you. This value is taken from the default value in the system settings. For this exercise leave this value unchanged.
- 4. Enter your name in the *From email display name*. If your class is using a common ToEmail Address for the Partner, this will make it easy to find your email.
- 5. For the *Email subject*, enter *Acknowledging receipt of your order*.
- 6. For the *Body text*, a sample file named **Ack.txt** will have been installed in the tutorial directory on your server. Enter the full path and file name for this sample file. For example,

#### /LANSA\_Composer\_licpgmlib/lic/Tutorial/Ack.txt.

If you are using Composer on a Windows server, use the prompt button to find the file.

7. Click *Save*.

📄 SMTP mail detail : Details (JMI	TUTEMAILACK-Acknowledging Receipt of your Order)	
Details 📋 Notes 🐼 Audit		
ID	JMITUTEMAILACK	Eave Save
Description	JMI Email Acknowledgement	Refresh
Status	Active	
To email address	to@lansa.co.uk	
CC email address		
Blind copy email address		
From email address	trainee11@lansa.co.uk	
From email display name	John Ivory	
Email subject	Acknowledging Receipt of your Order	
Body text file (location & name)	/lansa_composer_k12pgmlib/lic/tutorial/Ack.txt	
Attachment location & file		
Zip file attachment		

# Step 3. Add an Email acknowledgement to your Processing Sequence

You have set up configurations for the SMTP email. Now you can add an email acknowledgement to your Processing Sequence.

- 1. In the *Navigator*, expand *Definitions* and select *Processing sequences*.
- 2. Select your Processing sequence, iiiTUTSEQ03 and create a new processing sequence by copying it. Save the new one as ID: iiiTUTSEQ04, Description: iii Tutorial LIC006 and save it.

Use the *Edit* button to open the new processing sequence. You want to send a single email to a *Trading Partner* if any xml files were found and transformed for that Trading Partner.

- 3. From the *Palette* tab select the *If* construct and add it to the sequence at the end. Adjust, using *Promote*, the level of the **If**, so it is at the same level as the **FILELIST Loop**.
- 4. Select the *If* and in the *Condition* enter **&FILECNT \*GT 0**
- 5. From the *Activities* tab select **MAIL\_SEND** and add to the sequence after the **If** so that the **MAIL\_SEND** is within the **If**. You can adjust the levels if necessary using the *Promote/Demote* icons.
- 6. Select the **MAIL\_SEND** *Activity*. In the *Parameters* (bottom right), enter a *Variable or value for parameter* for SMTPMESSAGEDETAILID of the SMTP mail detail configuration you created in the last step (iiiTUTEMAILACK) or drag and drop it from the *Configurations* tab.
- 7. Add the *SMTP server configuration* (iiiTUTSMTPID) you created to the *Variable or value for parameter* for SMTPSERVERID.
- 8. Enter a *Variable or value for parameter for parameter TOADDRESS* of **\*TRADINGPARTNER.EM** (that is **\***TRADINGPARTNER without an "S" plus a full stop "." and **EM**)., or drag and drop it from the *Builtins* tab.

This means when the *Processing Sequence* is executed it will assign the value of the email address from the *Trading Partner* to the *TO* email address used by the MAILS\_END Activity. In this way the email will be sent to the *Trading Partner* being processed.

	□ <del>(</del> +) □ :	If Condition: <b>Activity</b>	&FILECNT > 0		
		Name:	MAIL_SEND - Send email by S *** End of processing sequence	MTP ce ***	
Deta	ils Parameter	8			
		Variable or va	lue for parameter	Parameter Na	Parameter description
1	<ul> <li>Inbound</li> </ul>	<b>UMITUTEMA</b>	ILACK'	SMTPMESSAG	SMTP mail details configuration II
2	Inbound	UMITUTSMT	P'	SMTPSERVER	SMTP mail server configuration IE
3	Inbound	*tradingpartne	er.em	TOADDRESS	Override TO address on detail co
4	Inbound			CCADDRESS	CC address
5	Inbound			BCCADDRESS	BCC address
6	Inbound			SMTPSUBJEC1	SMTP mail subject
7	Inbound			MAILBODYTEX	Body text path and file name
8	Inbound			MAILATTACHM	Path and file name for mail attach
9	Inbound			MAILATTACHM	File name of zipped attachment

You need to execute the MAIL\_SEND Activity if any entries were found in the FILELIST. Remember that FILECNT is the index for the FILELIST.

9. You now need to ensure that the FILECNT is always zero before building the FILELIST. You do not want the FILECNT to have a value left over from a previous Trading Partner. So you must set FILECNT to zero before the list is set up. To do this, select an *Assign* from the *Palette* and add to the sequence before the Activity DIRECTORY\_LIST.

10. Select the *Assign* and set the *Expression* to **&FILECNT = 0** 

Your processing sequence should now look like the following:



11. *Save* your changes to the processing sequence.

#### Step 4. Execute your Processing Sequence

- 1. Click the *Run* button.
- 2. Review the *Processing Log*.
- 3. Review the data in tables TUTORDH and TUTORDL.

Check the email account you defined for your *Trading Partner*. You will receive an acknowledgement email. The text of the email will be taken from the Ack.txt file.

#### Summary

- You have used more of the conditional constructs available within the *Processing Sequence*.
- You have defined *SMTP configurations* and used the MAIL\_SEND activity to send an acknowledgement email to a *Trading Partner*. This is just one of the configurations that are available. Other configurations and standard *Activities* available for use are *FTP*, *POP3* and *HTTP*.
- Setting up a configuration enables you to re-use the configuration structure within multiple *Processing Sequences* or for multiple *Trading Partners*. Various elements of the configurations may be overridden or varied when executed within a *Processing Sequence*.

## LIC007 - Extract Database to CSV File

#### Objectives

To extract a CSV file from a physical file, with selection criteria. You will create a transformation map and use it in a processing sequence to do the following:

- Extract employee data from the Personnel file (PSLMST) into a CSV file
- Select records based on department code
- Input department code as a run time parameter to the processing sequence
- Input the output CSV file name as a run time parameter

#### Steps

To achieve these objectives you will complete the following:

- Step 1. Create Transformation Map
- Step 2. Define Transformation Map
- Step 3. Define an SQL Where condition and input parameter
- Step 4. Prepare Transformation Map
- Step 5. Create Processing Sequence
- Step 6. Run the Processing Sequence
- Step 7. Run Composer CL Command (Optional)
- Step 8. Run Processing Sequence from a shortcut (optional)
- Summary

This exercise uses files from the LANSA Personnel Demonstration System.

On an IBM i server, the files are in the LANSA Composer partition library. In an installation using default names, the library name will be LICLICLIB.

On a Windows server, the files are in the designated LANSA Composer database, in schema or collection name LC\_DTA.

## **Before You Begin**

• In order to complete this tutorial, you should complete the preceding tutorials.

#### **Step 1. Create Transformation Map**

- 1. In the Navigator, select Transformation Maps.
- 2. Use the B wew button on the toolbar to create a new *Transformation Map* defined as:
  - ID iiiTUTMAP02

Description Employee File to CSV with Selection

Map Type Unspecified Type

Restartable Yes

Status Active

3. Save it

Your new *Transformation Map* details will look like the one below:

🔥 Transformation map : D	etails (IIITUTMAP02-Employee File to CSV with Selection - LIC007)	2
Details 🛛 🍭 Data intercha	ange   😫 Parameters   🔃 Run history   🔍 Where used   🛄 Notes   🐼 Audit	
ID	IIITUTMAP02	
Description	Employee File to CSV with Selection - LIC007	
Мар Туре	Unspecified type	
Restartable Yes 💌		
Status	Active	
Click Edit t	to edit your transformation map. Edit	
When you have saved your transformation map, click Prepare to complete preparing it for use. Prepare to complete preparing it for use.		
La cara antina a antin		

#### **Step 2. Define Transformation Map**

In this step you will add the file PSLMST – Personnel to the map and add a text file (CSV) and define its fields. You will then define the basic mapping between the two objects.

- 1. From the *Details* tab, use the *Edit* button to open *Mapforce*.
- 2. In *Mapforce*, use the *Insert / Database* menu option:

😂 Altova Mapf	orce - [IIITUTMAP02.mfd] 👘 🚽
🔁 <u>F</u> ile <u>E</u> dit	Insert Project Component Connec
🗅 🚅 🖬 🕼	ୱିସ XML Schema/File
🛃 📑 🕅 🗐 🗐	📭 Database
Libraries	EDI
Barrow	In Iext file

3. Select *IBM DB/2 (ODBC)* in the *Connection Wizard* and click the *Next* button.

😢 Select a Databa	se 🛛 🔀
Connection Wizard	<b>Connection Wizard</b> Please select a source database and click on next. If the database vendor that you wish to work with is not listed below, then please create a connection using either ADD or ODBC directly.
ADO Connections	<ul> <li>Microsoft Access (ADD)</li> <li>Microsoft SQL Server (ADD)</li> </ul>
	Oracle (ODBC) MySQL (ODBC) IBM DB2 (ODBC)
Global Resources	O Sybase ( ODBC ) O PostgreSQL (ODBC)
	Next > Close

4.On the next dialog, select ODBC:



5. If you already completed the earlier LIC003 tutorial exercise, then you will already have defined a DSN that you can again use here. In that case, select the 'Use an existing Data Source Name' option and follow the prompts to select the DSN you created earlier. Then proceed to step 8.

Otherwise, select the 'Create a new Data Source Name' option. Click the Connect button.



6. Enter a Data Source Name of <server>\_<library>, where <server> is the IBM i and <library> is the specific library which will be the default for this connection - this should be the library containing the Personnel Demonstration system files, for example, LICLICLIB.

Do not click *OK* at this point.

ries Access for Wil	ndows ODBC Setup
eneral Server Packa	ages Performance Language Catalog Translation Diagnostic
Data source name:	
EARTH_K12PGMLIB	
Description:	
iSeries Access for Win	dows ODBC data source
System:	
EABTH	Connection Options

7. Select the *Server* tab. Enter an SQL default library of <library> where <library> is the library containing the Personnel Demonstration system files, for example, LICLICLIB.

iSeries Access for Windows ODBC Setup								
General	Server	Packages	Performance	Language	Catalog	Translation	Diagnostic	
Namin	ig conver	ntion:						
SQL	naming c	onvention (*	SQL)	~				
SQL	lefault libr	ary:						
K12P	GMLIB							
Library	y list:							_

Then click OK.

- 8.Signon to the iSeries when the login dialog appears, using your iSeries user id and password.. You may need to signon three times (this depends on the configuration of IBM i Access security options).
- 9. In the Insert Database Objects dialog, the highlighted area shows the first IBM i library or the first database schema or collection. Select the arrow icon and select the required library, schema or collection from the drop down list.

As before, the required IBM i library will depend on the LANSA Composer installation, for example, LICLICLIB. On a Windows server, the schema name is LC\_DTA.



10. Select the file DEPTAB – Department Table.

🖷 Insert Database Objects 🛛 🛛 🔀
Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.
Source
▼ ▼ ♦ Add/Edit SELECT Statement Add/Edit Table Relations (0)
Image: State Stat
Show Preview >> Insert Cancel

11. Then scroll down and select file PSLMST – Personnel.

🛱 Insert Database Objects	
Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statement in the mfd file.	ments defined by the user are stored locally
Source	
🔻 🔽 💠 🛛 Add/Edit SELECT Statement 🕻 Add/Edit Table Relations (0)	
H I I Views	× 1
Show Preview >>	Insert Cancel

- 12. Select the *Add/Edit Table Relations* button (highlighted above).
- 13. In the *Add/Edit Table Relations* dialog, begin by selecting the *Add Relation* button.

2	Add/Edit Table Relations		
lr e d	this dialog you can define table relation ach relation a primary key table and a fo atatypes.	ns in addition to those defined in preign key table and at least one	the database. Please select for column pair with matching
	Primary/Unique Key Table	Foreign Key Table	ОК
			Cancel
			Add Relation
	and the second state of the second states		Delete Relation

14. In the left hand *Primary/Unique Key Table* column, select table DEPTAB and then column DEPTMENT. In the right hand *Foreign Key Table* column, select table PSLMST and then column DEPTMENT.

Add/Edit Table Relations		
In this dialog you can define table relations each relation a primary key table and a fore datatypes.	in addition to those defined in the databa eign key table and at least one column pa	ise. Please select for ir with matching
Primary/Unique Key Table	Foreign Key Table	ОК
🖵 🔟 DEPTAB 📃	🔟 PSLMST 📃 🔼	
1 DEPTMENT (CHARACTER(4 💌	🕕 DEPTMENT (CHARACTER 💌	Cancel
		Add Relation

15. Click the *OK* button.

16. Continue by clicking the *OK button* in the *Insert Database Objects* dialog. Your map should look like the following:



17. Add a text file using the Insert / Text File menu option.



18. Select the radio button – *Use simple processing for standard CSV or FLF files* and click *Continue*.



19. Ensure the checkbox First row contains field names is **NOT** checked. Initially there will be one field only defined. Click in the top line and replace Field 1 with EmployNumber.

	Input file	OK Cancel
Input / Output Enco	ding	
Encoding name:	Unicode UTF-8	
Byte order:	Little Endian	
First row contain:	s field names) s as absent	
Employflumber		1

- 20. Continue to add new fields by pressing the Append Field button and adding the following fields:
  - a. Surname
  - b. GivenName
  - c. Address1
  - d. Address2

- e. Address3
- f. PostCode
- g. Department
- h. DeptDescription
- 21.Change the field PostCode to Integer.
- 22.Check the checkbox First row contains field names .Your form should now look like the following:

I	"IC											-
L									Input	file		OK
									Outpu	t file	0	ancel
Save all file	paths relativ	e to MFD file										
Innut / Output F	Focoding											
Encoding name	: Unico	de UTF-8								•		
Byte order:	Little	Endian			+	l Ir	dude	byte ord	er mar	k		
CSV Settings Field delimiter	Semicolon (	Comma	) Spi	ace 🔘 O	Juston	1	T	extendo	sed in	o •	0	CSV
CSV Settings Field delimiter Tab 9 First row cor Treat empty	Semicolon ( ntains field n r fields as abo	© Comma ames sent	) Spi	ace 🔘 O	Juston	1	Tr C	ext endo ) Not (	sed in		e	CSV Fixed
CSV Settings Field delimiter Tab 9	Semicolon ( ntains field n r fields as ab	Comma     ames     sent     GivenNa	) Spa	ace © C	uston	Addres	T( ©	ext endo ) Not ( Addres	sed in	PostCo	@ © de	CSV Fixed
CSV Settings Field delimiter Tab S First row cor Treat empty mployNumber tring	Semicolon ( ntains field n fields as ab Surname string	© Comma ames sent GivenNa T string	Spi	ace © C Address string	luston	Addres	52 •	Addres	sed in	PostCo	e C de	CSV Fixed Depar string
CSV Settings Field delimiter Tab S First row cor First row cor Treat empty mployNumber tring	Semicolon ( ntains field n / fields as ab / Surname string	© Comma ames sent     GivenNa     string	© Spa ame ▼	ace © C Address string	luston	Address	s2	Addres	sed in )' ( :s3 	PostCo integer	e C de	CSV Fixed Depar string

23. *Save* your changes by clicking the *OK* button.

Your form should now look like the following:

🙄 Componen	t Settings											×
Input / Outpu	t File											
									Inp	ut file		ок
									Out	out file	C.	ancel
Save all fi	le paths relati	ive to MFD f	ile									
Input / Outpu	t Encoding											
Encoding nar	ne: Unico	ode UTF-8								~		
Byte order:	Little	Endian				Image: A state of the state	Inclu	de byte o	rder n	nark		
Field delimit Tab	er ) Semicolon :ontains field i ity fields as ab	⊙ Comma names osent	0	Space 🤇	) Cu:	itom	][	Text end	losed	in ©"	0	CSV Fixed
GivenName	Address1	Address	32	Address	3	Postcod	e	Departn	nent	DeptDes	cription	
string 💌	string 📘	string	-	string	•	integer	•	string	-	string	-	
<	Insert Field	t) (Roman	e Fiel	a			1111					

24.Click *OK* to insert the text file.

Your map should now look like the following:



25.Expand the DEPTAB table below table PSLMST and then define the required mapping between fields in tables PSLMST and DEPTAB and fields in the text file.

When mapping the DEPTDESC field, it is important that you select the instance of DEPTAB/DEPTDESC that is nested below PSLMST. This ensures that the database table relation you defined in step 14 is effective and that you pick up the DEPTDESC value that corresponds to the DEPTMENT (department code) that is in PSLMST.

Your map should now look like the following:



Mapforce will display a message box, because at this stage the input database file is not connected to *Rows* in the output text file. Click *OK* and continue. You will be completing this link in the next step.



#### Note ODBC DSN

In general *MapForce* database components are qualified with the database schema name. On IBM i, this corresponds to the IBM i library name.

In the context of *LANSA Composer*, it would not be satisfactory for the prepared *Transformation Map* to contain hard-coded references to database tables in a named library, which is what would happen if *LANSA Composer* did nothing else.

For LANSA Composer version 3.0, the map definition (the .mfd file) is modified

BEFORE MapForce generates the Java code. The modification (if found to be necessary) is made to a temporary copy of the map definition (.mfd) file that is used only for the *Prepare* step (the original map definition file is NOT changed). The nature of the modification is to automatically switch on MapForce's own "Strip schema names" option in the database component, and thereby rely on MapForce's own code generator to get the unqualified database references correct. (This MapForce option was not available in earlier MapForce versions.)

One other consideration is that "strip schema names" may lead to a "Prepare Map failure" due to the required file(s) not being found if:

- the map uses SQL SELECT and/or SQL-WHERE components, AND
- the ODBC DSN in use does not specify an appropriate library list.

This means that in such cases it will be necessary to use a DSN that specifies an appropriate SQL default library and/or library list.

This and related issues are discussed and described more fully in *Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS* in Appendix C of the *LANSA Composer User Guide*. It is recommended reading if you work with *Transformation maps* that reference IBM i databases in *LANSA Composer version 3.0 (or later)*.

#### **Further note:**

Even when *MapForce's* own "Strip schema names" option is used (whether by the map designer or applied automatically by the *Prepare* step as described above), *MapForce* does NOT strip schema names from user-entered SQL - in other words, from any SQL entered using MapForce's SQL SELECT feature.

#### Step 3. Define an SQL Where condition and input parameter

This transformation map needs to support the selection of employees by department code to be inserted into the CSV text file. In this step you will complete the transformation map by adding an *SQL Where Condition*, and an input parameter linking the file PSLMST to the output text file via the *SQL Where Condition*.

1. Insert an *SQL Where Condition* using the *Insert / SQL-WHERE Condition* menu option.



2. Link the PSLMST file to the table element of the *SQL-Where*, this will open the *SQL-Where* properties dialog:



3. Define the selection as DEPTMENT = :Department\_In (note the colon - and lack of space - before Department\_In). This will define an input parameter – see example below.

**Note:** DEPTMENT is the name of the department code field in the file PSLMST, which the transformation is reading.

SQL-WHERE	Properties	×
To create a para Sample: Name = SELECT (fields) I	meter, enter a <sup>(</sup> ) followed by its name in the WHERE condition. ::Name AND Age >= :ValidAge FROM D11LICLIB.PSLMST WHERE	
DEPTMENT = :	Department_In	<
Parameters:		~
Parameter	Туре	
Department_In	(auto-detect by database)	•
1	ОК Са	incel

4. Click the *OK* button to save these changes. Your map window should now look like the following:



5. Using the *Function / Insert Input* menu option to define an Input parameter.



6. Define a string input called Department\_In and click *OK*.

**Note:** You can also define a default value and use this in *Mapforce* to produce sample output.

Create Ing	put 🛛 🔀
Name:	Department_In
Datatype:	string 🔽
	Input is required
Design-tir	me Execution
🔽 Speci	fy value
Value:	ADM
	OK Cancel

7. Link the *Input* function to the *SQL-WHERE* Condition and link the *SQL\_Where result* output to the text file *Rows*, as shown below.



8. Use the Save icon to save your transformation map. Note that the messages area at the bottom left of Mapforce should look like the following:



9. Select *Output* tab. Provided you specified a default value in the Input parameter, this will show the employees which will be selected. **Note:** this view has been scrolled to the right, to show the Department Description column which has been retrieved from the related table DEPTAB.

1	Address3,PostCode,Departm	ent,DeptDescription			
2	","6 Camillo Avenue	", "SEVEN HILLS.	", "NT	",2147,"ADM ","ADMINISTRATION	
3	","12 Augusta Avenue,	", "PUNCHBOWL.	","QLD	",2016,"ADM ","ADMINISTRATION	
4	","2 Burton Road,	","LANE COVE.	","VIC	",2100,"ADM ","ADMINISTRATION	
5	","59 Darley Road,	", "BEXLEY.	", "TAS	",2030,"ADM ","ADMINISTRATION	
6	","6 Reading Avenue,	", "KINGSLANGLEY.	","NSW	",2147,"ADM ","ADMINISTRATION	
7	","15 Baker Place,	", "PENSHURST.	","WA	",2153,"ADM ","ADMINISTRATION	
8	","14 Whitby Road,	","ST IVES.	","SA	",2020,"ADM ","ADMINISTRATION	
9	","47 Lincoln Street,	","STANMORE.	","WA	",2007,"ADM ","ADMINISTRATION	
10	","1 Main Rd	", "Hill Top	","ACT	",2345,"ADM ","ADMINISTRATION	
11	","22 Moton Street	","Marrickville	","WA	",2090,"ADM ","ADMINISTRATION	
12	","122 Arthur Street	", "North Sydney	","ACT	",2060,"ADM ","ADMINISTRATION	
13	","x	","x	","QLD	",2000,"ADM ","ADMINISTRATION	
14	","x	","x	","WA	",2000,"ADM ","ADMINISTRATION	
15	","x	","x	", "WA	",2000,"ADM ","ADMINISTRATION	
16	","x	","x	","SA	",2000,"ADM ","ADMINISTRATION	
17	","x	","x	","x	",2000,"ADM ","ADMINISTRATION	
18	","x	","x	","x	",2000,"ADM ","ADMINISTRATION	
19	","x	","x	","x	",2000,"ADM ","ADMINISTRATION	
20	","X	","X	","X	",2000,"ADM ","ADMINISTRATION	
21					
1-2 ·	and share and and a second second	a second and a second	and a	and a second	
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		C		1. A. A	

10.Close *Mapforce*.
#### **Step 4. Prepare Transformation Map**

1. Use the Prepare Prepare button on the Details tab to prepare the Java service on the IBM i which will implement your transformation. At the *Prepare Transformation Map* window, click the *Prepare* button to confirm your request:

2rs	Transformation map	IIITUTMAP02					
35	Description	Employee file to CSV with Selection -	Employee file to CSV with Selection - LIC007				
	transformation map. Click Prepare to proceed.  Cance Cance						
Check Prepro Gener Compi	Folders and files poess transformation map defir ate code for transformation map le code for transformation map igue information for transforma	iition ap tion map					

You may see the MapForce window open briefly during the Prepare – this is normal and it will close by itself in a few moments. If prompted, remember to complete the *IBM i Signon* screen during the *Prepare* steps, which may not be shown as the top window.

2. When this step completes – and it may take a few minutes, you will see the following message boxes:



When the remaining steps have completed, click *Close* to close the *Prepare Transformation Map* window.

3. The jar file for your transformation map has been created and placed on the

server in a folder such as:

/lansa\_composer\_licpgmlib/jsm/instance/composer/LIC/map Where: licpgmlib is your *LANSA Composer* program library And LIC is the LANSA partition which Composer is using.

## **Step 5. Create Processing Sequence**

In this step you will create a simple processing sequence which will run your file transformation. It will have two input parameters, Department Code and output file name.

Before you begin:

- a. Create a new *Database Configuration* (iiiTUTDEMO), similar to the one you created in LIC003, Step 7. *Set up the Database Configuration*.
- b. Copy the Database Configuration **iiiTUTDBID.**
- c. Give the new *Database Configuration* a suitable *Description*.
- d. If necessary, change the *Connection string* to use the library containing the Personnel Demonstration System files (for example, LICLICLIB).

Note that you must specify the *Database password*. Security details are not copied.

- e. *Save* your new database configuration.
- f. In the *Navigator* select *Definitions / Process Sequences*. Use the button to create a new processing sequence. This will immediately open the *Processing sequence editor*.



2. Select the *Transformations* tab from the top left hand tabs, and drag and drop your transformation iiiTUTMAP02 at the end of the processing sequence in the right hand *Details* tab – see below.

Note: the Parameters tab (bottom right) has focus and shows the three

parameters required by this transformation. They are:

- EARTHSourceConnection the name of the DB Source
- Department\_InSourceParameter the department code for which employees are to be selected
- TextfileTargetFileName the name (and path) for the output CSV file.



3. Select the *Configurations* tab (top left). Drop the iiiTUTDEMO *Database configuration* which you just created into the EARTHSourceConnection parameter.

Ontitled" (Untitled) - Processing sequence editor	
Elle Edit Yiew Insert Help	
Save 💾 🛛 Run 🕨 🎂 💥 🔏 🗊 🖺 🛷 Ei	inabled 🛛 🤤 Promote 🆙 😓 Demote 🔿
🤗 Palette 🛛 🌦 Activities 🛛 🌠 Processing sequences	Details 🚽 Parameters 🚺 Notes
😳 Built-ins	Item Details
in Transformations 😵 Configurations 🤴 Variables	**** Beginning of processing sea
8 0 Search	🖃 🍪 Transform
	Name: IIITUTMAP02 - Employee File t
Name Description	*** End or processing sequence
Configurations by group     Database configuration	
EXAMPLEDB 400 Example System i dat	
EXAMPLEDB_ACCES Example Microsoft Ac	
DOMPLEDB_MSSQL Example SQL Server	
EXAMPLEDB_ODBC Example database co	
EXAMPLEDB_ORACL Example Oracle datab	
IIITUTOBID II System i database c	
IIITUTDEMU III System i database c	
FTP configuration (Command	
FTP configuration (command	
	Details Carameters
Assistant 😽 Errors	Variable or value for parameter Par
	1 * Inbound 'IIITUTDEMO'
Value for parameter IIITUTMAP02_2 is expected by Transform	2 * Inbound
Value for parameter IIITUTMAP02_3 is expected by Transform	3 • Inbound IIII

- 4. In this step you will define two input parameters for the *Processing Sequence*.
  - a. Select the *Variables* tab (top left) and select the *Parameters* tab (top right).
  - b. Drag and drop the variables iiiTUTMAP02\_2 and iiiTUTMAP02\_3 onto the *Parameters* tab. You can re-sequence them using the **Arameters** toolbar buttons if necessary.



- 5. Give each of them appropriate descriptions as shown (these are displayed at run time). Select the *Parameter Name* iiiTUTMAP02\_3 and rename it to FILENAME. **Note:** you could also provide a default value for each parameter.
- 6. Select the *Activities* tab (top left) and the *Details* tab (top right). Drag and drop the PATHMAKE activity into the processing sequence before the

Transform. You can re-sequence using the *Promote*  $\bigcirc$  *Demote*  $\Rightarrow$  buttons if necessary.

Processing sequences	📔 Details 🛛 🚽 Parameters			
Transformations	Item	Details		
🔗 Configurations 🛛 🧐 Variables		*** Beginning of processing sequence ***		
Built-ins	E / *	The PATHMAKE active enables a path and	file name to be defi	ned.
2 Palette Activities	*	The variable output by PATHMAKE is used	for the output CSV	path/ file name for the Transformati
	🖃 🍰 Activity			
All activities 🔽	Name:	PATHMAKE · Makes a file path from path and file	e name	
	🖉 💼 Transform			
Activity ID Activity d	Name:	JMITUTMAP02 - Employee File to CSV - LIC007		
MSG_SEND Send file.		*** End of processing sequence ***		
REXTNUMB Generat				
📸 NOTIFYEVE Notify E				~
NULL /ull acti	Details Parameters			
PATHMAKE Makes a	Mariable or us	luo for poromotor	Parameter Marea	Parameter description
🚔 PATHSPLIT 🛛 Splits a fi 🔄	1 * Inhound		PATHIN	Input path
🍰 POP3MAIL Get all e 📒	2 Inhound		FILEIN	Input file name
🍰 POP3MAIL Get singl 🦳	3 Inhound		EXTENSIONIN	Input file extension
🍰 PUT_DTAARA Write a	4 Outbound		PATHOUT	Resulting full file path
🚔 RANDOMNU Generat				
JAME FILE Berman		man man man	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	man man

7. Review the *Assistant* tab for the PATHMAKE activity. These notes explain that the activity will use the correct separator depending on server platform (Windows or System i) and will insert a separator if not specified. In the next steps you will set up parameters for the PATHMAKE activity.

# 8.Define the PATHIN inbound parameter as

'/LANSA\_Composer\_licpgmlib/lic/Tutorial/'.

**Note:** if you do not enter the string within single quotes, they will be added when you save the changes.

The actual path for the training system settings which you are using can be found via the *Navigator*, under *Administration & Housekeeping / System Settings / Tutorial Files*.

Note: the *Processing sequence editor* is non-modal. This means you can switch back to the main Composer window to review this information, leaving the Processing sequence editor open.

9. Define the FILEIN inbound parameter as &FILENAME. This is the input parameter you renamed earlier. It contains the *Output File Name*.10.Select the *Variables* tab (top left) and drag and drop the variable iiiTUTMAP02\_3 into the outbound PATHOUT parameter of the PATHMAKE activity. This will create /path/filename variable iiiTUTMAP02\_3 which is required in the *Transform output file* parameter.

🤔 Palette   🍰 Activities	📄 Det	ails 🚽 Pa	arameters 🔲 Notes		~
Processing sequences	Item			Details	5
Constructions Transformations				**** Beginning of processing sequence **	× )
Built-ins	B / 1	*		The PATHMAKE activity enables a	path and file
Configurations Variables	-	Activity		The output variable from PATHMA	KE is used a
	n	lame:		PATHMAKE - Makes a file path from pat	h and file name
EXTENSIONIN     FILENAME     FILENAME     IIITUTMAP02_1      IIITUTMAP02_2	۱ 🚵 🗆 ۱	Transform Name:	-	IIITUTMAP02 - Employee File to CSV wil *** End of processing sequence ***	th Selection
PATHIM	Details	Parameter	\$		
			Variable or value for para	ameter	Parameter Na
	1.	Inbound	VLANSA_composer_L35PGM	ILIB/lic/Tutorial	PATHIN (
	2	Inbound	&FILENAME		FILEIN
Assistant 🐨 Errors	3	Inboard			EXTENSIONIN
	4	Outbound	WIITUTMAP02_3		PATHOUT
PATHMAKE	<b></b>	(			m s

10. Drag and drop a *comment* from the *Palette* tab, before the PATHMAKE activity and add suitable comment text in the multi-line edit box below. See example:

📔 Details 🚽 🚽 Parameters		
Item	Details	
	*** Beginning of processing sequence ***	
□ / *	The PATHMAKE active enables a path and file name to be defined.	
*	The variable output by PATHMAKE is used for the output CSV path/ file name for th	
🖃 🊔 Activity		
Name:	PATHMAKE - Makes a file path from path and file name	
🖃 💼 Transform		
Name:	JMITUTMAP02 - Employee File to CSV - LIC007	
	*** End of processing sequence ***	
		$\sim$
Comment:		
The PATHMAKE active enab	iles a path and file name to be defined. HMAKE is used for the output CSV path/ file name for the Transformation	

- 11. Add an *UPPERCASE* activity before the *Transform Map*. Set its input and output parameters to IIITUTMAP02\_02, which is the input department code. Since the file extract uses SQL, this would fail to output to the CSV file if the user enters the department code in lower or mixed case.
- 12. Click on the Save Save button to save your processing sequence. Complete the form with name (iiiTUTSEQ05), description and Status set to Active.

Save	
Type name an	d description for new processing sequence:
Name:	IIITUTSEQ05
Description:	iii Employee to CSV - LIC007
Status:	Active
Save	Cancel

#### Step 6. Run the Processing Sequence

1. Run the processing using the Run Run button. Enter the required parameters *Department* (for example **ADM**) and *Output File Name* (for example iiiEMPCSV.csv). Then click the *Run* button to run your processing sequence.

Þ Run Prod	cessing Sequence		
<b>**</b>	Processing sequence Description	JMITUTSEQ0 Employee to	05 CSV - LIC007
	O Submit to <u>b</u> atch	۲	Run immediately
	Type parameter val	ues for this run:	
	Department		
	Output File Name		
V Close	R <u>u</u> n	Cancel	<u>S</u> ave shortcut

2. If you have set up the processing sequence successfully, your log should look like the following.

200	Processing sequence	IIITUTSEQ05		
<b>1</b>	Description	iii Employee to CSV - LIC007		
	Run number	0000000000721		
	Start date / time	24/02/2009 17:25:18		
	End date / time	24/02/2009 17:25:19		
	Execution result code	OK 💌		
	Processing sequer     Parameter value	nce started	Timestamp: Message ID: Message type:	24/02/2009 17:25:19 DXS0110 *COMP
	Completed act     Completed act     Activity sta	ivity CONCAT Irting: Employee File to CSV with Selec ansformation map ended	Message text:	Elapsed (seconds): 1.76
	Processing ending			

3. Use Windows Explorer to find your output CSV file. You will need to be signed into the IBM i server in Explorer to do this.

😂 Tutorial					
File Edit View Favorites Tools Help 🧗					
🕒 Back 🝷 🕥 🚽 🏂 🔎 Search	Polders 🕼 🍞 🗙 🍤 💷				
Address 🛅 \\earth\D11PGMLIB\LANSA_Com	er\Tutorial	💌 🄁 Go			
Folders	Name 🔺 Size Type	Date Modified 🔼			
Nokia Phone Browser	Bemp_out.csv 0 KB Microsoft Office Exc	18/07/2008 13:21			
My Network Places	Employee.csv 1 KB Microsoft Office Exc	21/11/2007 16:33			
Entire Network	Employees.dtd 1 KB DTD File	21/12/2007 16:41			
🗉 🚯 Microsoft Terminal Services	Employees.xsd 3 KB XSD File	21/12/2007 16:40			
🖃 👩 Microsoft Windows Network	Beugene.csv 1 KB Microsoft Office Exc	01/10/2008 16:17			
🗉 💑 Lansauk	F5_TUT_01_P1.txt 1 KB Text Document	12/11/2007 14:11			
🗉 🎿 Workgroup	ILEMPCSV.csv 3 KB Microsoft Office Exc	25/02/2009 09:52			
🗉 💑 Unknown	iseq01.xml 0 KB XML Document	18/07/2008 13:12			
🖃 🛃 earth	JMI_CSV7.csv 3 KB Microsoft Office Exc	11/12/2007 14:27			
🗉 🤶 COMPOSER	JMI_EMP_DET6.csv 3 KB Microsoft Office Exc	20/08/2008 15:52			
E 👮 D11HTDOCS	JMI_TUT_01_P1.txt 1 KB Text Document	11/12/2008 09:57			
when the standard and the standard and	Constant of the second of the	happener hand and a			

4. Double click to open this file in Excel. Your file should look like the following:

× 1	Microsoft E	xcel - JMI_EMF	P_DET1103.cs	۲.					
:2	Eile Edit	<u>V</u> iew <u>I</u> nsert	Format <u>T</u> ools	<u>D</u> ata <u>W</u> indow	Help Adobe PDF				
:	📔 🔒 🔒	) 🖪 💁 🗇	🕰   X 🗈	🔁 = 🍼   🔊 = C	- 😣 Σ - Å	A   🛄	43 📀	🚆 Arial	- 10 - B
	• • • •	0 🗞 🏹 🖉	ð 🖻 🖪 (	🖻   🚧 Reply with Ch	anges End Review	v			
: @	Snagit 🛃	Window	• -						
	J3	▼ fx							
	A	В	C	D	E	F	G	Н	
1	EmployNu	Surname	GivenName	Address1	Address2	Address3	PostCode	Department	DeptDescription
2	A1012	PAUL	PATRICK	6 Camillo Avenue	SEVEN HILLS.	NT	2147	ADM	ADMINISTRATION
3	A1013	PATTISON	GEORGE	12 Augusta Avenu	PUNCHBOWL.	QLD	2016	ADM	ADMINISTRATION
4	A1014	MOORE	JOHN	2 Burton Road,	LANE COVE.	VIC	2100	ADM	ADMINISTRATION
5	A1015	WOODS	BRADLEY	59 Darley Road,	BEXLEY.	TAS	2030	ADM	ADMINISTRATION
6	A1020	DOUGLAS	ADAM PETE	6 Reading Avenue	KINGSLANGLEY	NSW	2147	ADM	ADMINISTRATION
7	A1021	MCCULLY	DAVID	15 Baker Place,	PENSHURST.	WA	2153	ADM	ADMINISTRATION (
8	A1025	ROBINSON	MARY	14 Whitby Road,	ST IVES.	SA	2020	ADM	ADMINISTRATION
9	A1027	MORRISON	ALAN	47 Lincoln Street,	STANMORE.	WA	2007	ADM	ADMINISTRATION
10	A1111	VEREY	WARREN PE	1 Main Rd	Hill Top	ACT	2345	ADM	ADMINISTRATION
11	A1404	MRS BRICK	GILL	22 Moton Street	Marrickville	WA	2090	ADM	ADMINISTRATION
12	A1509	REDFORD	ROBERT	122 Arthur Street	North Sydney	ACT	2060	ADM	ADMINISTRATION
13	A2222	JOE	BROWN	х	х	QLD	2000	ADM	ADMINISTRATION
14	A3333	SUSAN	JAMES	X	X	WA	2000	ADM	ADMINISTRATION

You have completed this exercise. Following is an optional exercise if you are using an IBM i server.

# Step 7. Run Composer CL Command (Optional)

You can complete this step if you are using an IBM i server. On the *Run Commands* tab, the *Processing Sequence* creates the COMPOSER CL command string which can be used to execute the processing sequence directly on the System i. For example it could be placed in a CL job or it could be used to call the processing sequence from an RPG program.

**Note:** if you are a LANSA developer, there is a *Composer* program interface available from an RMDL program. See the *Composer online guide* for more information.



- 1. Start a 5250 emulator session and logon with your training course IBM i user id and password. Use CALL QCMD to bring up the command entry screen. Use F11 to make the command line full screen.
- 2. Select the *Run commands* tab shown above for processing sequence iiiTUTSEQ05. Select the command and paste it into your 5250 emulator screen using the emulator *Edit/Paste* menu option:

🕌 1 - EARTH - newlook
<u>Eile Session Edit View Macros Iools Keyboard Window H</u> elp
ØØ⊖ X ħĒØ ■⊇ ¤ ØØ ØØ Ø ₩ ₩ Ø
Command Entry
Type command, press Enter. ===> <u>D</u> 11PGMLIB/composer request(RUN)
/* pseg(IIITUTSEQ05) */
pseq(522431DE41F04E8F96612EF0CE94C2E7)
parms( (DEPARTMENT ' <parameter-value>')</parameter-value>
(FILENAME ' <parameter-value>') )</parameter-value>
partition(LIC) language(ENG)

3. Press F4 to prompt the COMPOSER command:

😹 1 - EARTH - newlook
Elle Setsion Edit View Macros Iools Keyboard Window Help
ØØ # 16 0 = = = \$ # # 4 # \$
LANSA Composer (COMPOSER)
Type choices, press Enter.
Function requested > RUN RUN. LICENSE. LICENCE
Processing sequence ID > 522431DE41F04E8F96612EF0CE94C2E7
Parameters:
Parameter name > <u>DEPARTMENT</u>
Parameter value > <u>'<parameter-value>'</parameter-value></u>
Parameter name
Additional Parameters
Partition > <u>LIC</u> Character value Language > <u>ENG</u> Character value

- 4. Set DEPARTMENT and FILENAME parameter values as appropriate and press enter to run the COMPOSER command which will run your processing sequence.
- 5. Display the output CSV file from the ....\LANSA\_Composer\Tutorial folder.
- 6. Note that you can review the run log from the *Composer Console*.

#### Step 8. Run Processing Sequence from a shortcut (optional)

1. Use the *Run* button to start processing sequence iiiTUTSEQ05. Note that it contains a *Save Shortcut* button.

Run Processing Sequence								
1	Processing sequence Description	JMITUTSEQ05 Employee to CSV - LIC007						
	Submit to <u>b</u> atch							
	this run:							
	Department							
	Output File Name							
Close	Run C	ancel Save shortcut						

- 2. Click the *Save shortcut* button.
- 3. Select Desktop and click the Save button



Note the *Processing sequence shortcut succesfully saved* message..



4. Switch to the desktop. Click the *Composer Run* icon.



5. Run the iiiTUTSEQ05 processing sequence. Note that you will be prompted to logon and the usual processing sequence run dialog will be shown.

The *LANSA Composer* client must be installed on the desktop in order to run a processing sequence shortcut.

#### Summary

- You have created a transformation map to extract records from a physical file, into a CSV format file.
- You defined an SQL Where condition and input parameter for the transformation map.
- You defined a processing sequence to run the transformation map, with two input parameters for department code and output file name and used the CONCAT activity to create the output path and file name information for the transformation.
- Optionally, you have used the COMPOSER command on the IBM i to execute your processing sequence directly on the IBM i server.
- Optionally, you have created a processing sequence shortcut on the desktop and run it.

# LIC008 - Processing via Email

# Objectives

To extend the previous exercise to do the following:

- retrieve an email request
- re-use the transformation map iiiTUTMAP02 to produce a CSV file extract from the Personnel File using information from the email request
- To send an email response with the CSV file as an attachment

The email subject line will contain a department code, which will be used by the transform (Employee file to CSV file) to create a CSV file of employees in the department.

This type of processing sequence could be used to handle many different types of inquiry via email. For example, a product enquiry to return price, stock availability or specification information.

This tutorial requires two email addresses:

- Send a request from your own email account, with the subject line containing a department code
- Send the request to another email address. The processing sequence will use the POP3\_MAILSINGLE activity to retrieve and delete all emails from this email account's inbox. You will probably need to set up this second email account.

Retrieve from your own email account. The MAIL\_RECEIVE activity will retrieve a single email message from the inbox. You will need to send a request to this email account.

# Steps

To meet these objectives you will complete the following:

- Step 1. Review Folders and Settings
- Step 2. Create POP3 Email Configuration
- Step 3. Create SMTP Mail Details
- Step 4. Define iiiTUTSEQ06 Processing Sequence
- Summary

# **Before You Begin**

• Before completing this tutorial, you should have completed all the preceding

tutorials.

## **Step 1. Review Folders and Settings**

1. Use Windows Explorer to display the folder:

\\<iSeries>\LANSA\_Composer\_LICPGMLIB\LIC\Tutorial\ Where:

<iSeries> = the training IBM i domain name

LICPGMLIB = the name of the *LANSA Composer* program library

LIC = the name of the LANSA partition which Composer is using Your folder should look like the following:

🗅 Tutorial 📃 🗖 🔀								
File Edit View Favorites Tools Help			<b></b>					
G Back ▼ (2) ▼ (2) Search  Folders  Search  Search  Folders  Search  Search								
Address 🛅 \\earth\k12composer\lic\Tutorial			🕶 🔁 Go					
Folders ×	Name 🔺	Size Type						
🖽 😼 A7800709 🛛 🔼	CEmailSkeletons	Folder						
표 冕 A7800710	C _ReceivedEmail	Folder						
🖃 😼 earth	CSentEmail	Folder						
🗉 🧟 AXESD11SCREN	DMI_Archive	Folder						
🗉 🧟 AXESD115KINS	🛅 JMI_EmailAttch	Folder						
🗉 🧟 AXESSLSSCREN	🛅 JMI_EmailSkels	Folder						
🗉 🧟 AXESSLSSKINS	🛅 JMI_EmailSubscriberAttch	Folder						
🗉 🧟 AXEST11SCREN	🛅 JMI_SentEmail	Folder						
🗉 🧟 AXEST11SKINS	DMI_Store01_Remote	Folder						
🗉 🧟 Composer	DMI_Store02_Remote	Folder						
Ⅲ Q D11HTDOCS	DMI_Store_Local	Folder						
🗉 🤶 D11PGMLIB	🛅 JMIFTPIN	Folder						
E 🧟 D12PGMLIB	DMIFTPOUT	Folder						
🐨 🖉 INTRANET 🔂 🛅 JMIReceivedEmail Folder								
🖃 🤶 k12composer 📃	🗐 Ack.txt	1 KB Text Doc	ument					
🖃 🦳 lic	FS_TUT_01_P1.txt	1 KB Text Doc	ument					
			h					

**Note:** the path shown in Windows will depend on the File Shares defined for the IBM i server. In the above example, the share  $\ensuremath{k12composer, maps to \ensuremath{ansa_composer_k12pgmlib.}$ 

2. In this exercise you will use these folders:

- \\_EmailSkeletons will contain a text file which is used to produce the body of the reply email.
- \\_SentEmail will be used for output and will contain the reply email text and Employee CSV file.

If these folders do not already exist, create them.

3. You will also need to create folder **\iii\_ReceivedEmail**, using your initials. It will be used to save the email message received via POP3.

## **Step 2. Create POP3 Email Configuration**

- 1. Navigate to Configurations / POP3 email configurations.
- 2. Select the EXAMPLE\_POP3 configuration and use the *Copy* button to create a new POP3 Configuration. This is set up to retrieve email from an external email inbox and save email text and attachments. It removes retrieved email from the server after it is read.



3. Enter or change the following information using your initials in place of iii:

ID	iiiTUT_POP3
Description	iiiTutorial POP3
POP3 Mail server	***
POP3 MailServer user profile	***
POP3 MailServer password	***
Save Directory	/LANSA_Composer_licpgmlib/lic/Tutorial/iii_ReceivedI
Status	Active

\*\*\* Enter your email account, user id and password.

Details 🚺 Notes 🕢 Audit					
ID	JMI_TUT_POP3				
Description	JMI POP3 mail configuration				
Status	Active				
POP3 Mail server	jupiter.lansa.co.uk				
POP3 MailServer user profile	trainee11@lansa.co.uk				
POP3 MailServer password	*****				
POP3 port					
Use secure connection (SSL)	No				
Public key name	{				
Public key password					
Save mail text	Yes				
Save mail attachments	Yes				
Save directory	/lansa_composer_k12pgm/lic/tutorial/JMIReceivedEmail				
Remove from server after read	Yes				

4. Save your changes

### **Step 3. Create SMTP Mail Details**

In this step you will define details of the SMTP response email your processing sequence will send.

- 1. Navigate to *Configurations / SMTP* mail details and use the button to define the SMTP Mail Details using the following settings:
- **ID** iii\_INQ\_EMP

Description	iii Tutorial Employee List Response
Status	Active
To email address	Use your own email address if accessible via the web*
From email address	Use your second email account.*
From email Display Name	iii Personnel Services
Email subject	Tutorial Employee Department List
Body text file	'/LANSA_Composer_licpgmlib/lic/Tutorial/iii_EmployeeInfo.txt'

📄 Details 🛛 🗞 Cross references 🛛 🖉 Attachments 🛛 💭 Notes 🕢 Audit								
Name	III_INQ_EMP							
Description	Employee List Response							
Status	Active							
To email address	john.ivory@lansa.co.uk	]						
CC email address								
Blind copy email address								
From email address	trainee11@lansa.co.uk							
From email display name	John Ivory							
Email subject	Tutorial Employee Department List							
Attachment								
Zip attachment as								
Body text file	/LANSA_Composer_licpgmlib/LIC/Tutorial/iii_EmployeeInfo.txt							
Content type								
Encoding								

**Note:** Once you have created your text file, you may be able to use the Edit icon against the *Body text file* parameter to open the file in Notepad, if the file already exists and if the file path specified is on the LANSA Composer "home" path.

2. Save your *SMTP Email Details* definition

## Step 4. Define iiiTUTSEQ06 Processing Sequence

In this step you will create a processing sequence to do the following:

- Assign assign variable = OK
- While loop while variable = OK
  - Activity Get single email from server via POP3
  - If &FROMADDRESS \* NE "
    - Transform Extract Employee file to CSV
    - Assign &TONAME = 'Personnel Dept LANSA'
    - Activity text substitute build email body
    - Activity Send email response via SMTP
  - Else
  - Terminate

First you will create a text skeleton which will later be used by a text substitute action to create the response email body text.

1. Open Notepad and create the response text skeleton shown below:



2. Save this to the IBM i IFS as:

#### Notes:

- Notepad will automatically add a txt filename extension
- This tutorial path can be found under *Administration & Housekeeping / System settings / Tutorial files*
- The value %%PARM.TONAME%% and %%PARM.DEPARTMENT%% will be replaced by the *Text Substitute* action you will include in the

processing sequence.

- 3. On the *Navigator panel* select *Definitions / Processing sequences*. *Click* the button to create a new *Processing Sequence*. The Processing sequence editor will open immediately.
- 4. Start defining the processing sequence by selecting the *Palette* tab (top left) and dragging and dropping the *Assign* processing directive on to the *Details* tab. Assign the value 'OK' to a variable &FLAG\_WHILE. This will be used to control the While loop.



5. Next drop a *While* loop into the processing sequence following the *Assign* activity. Set the condition to &FLAG\_WHILE = 'OK'. The loop will later be terminated by an If condition.

🜍 Untitled* (	(Untitled) - Proc	essing sequence edit	or		- • •
Eile Edit 1	view Insert	Help			
Save 💾	Run 🕨	də 🗙 🕺 🚺 🛙	🖞 🛷 Enabled	🖨 Promote 🛛 🚭 😽 Demote 🖨	
💕 Processin	ig sequences	b Transformations	Details 🏼 🌓 Param	eters 🚺 Notes	
Configural	tions   🤭 Varia	bles 🛛 🥘 Built-ins	Item	Details	_
😵 Palette	Activities		and the second s	**** Beginning of processing sequence ***	
C	8	<b>(3</b> A	Expression:	8FLAG_WHILE = 'OK'	
Loop	While	Until	Condition:	&FLAG_WHILE = 'OK' **** End of processing sequence ****	
6	6	♠			*
Leave	Continue	Switch	Details:		
			Condition		
-		70.	&FLAG_WHILE = 'OK'		
Case	Otherwise	<b>{*}</b>	Index variable		
		<b>E</b>	Maximum iterations	999	
	a shine	R			

The *While* will be highlighted as an error since there is nothing inside the loop at the moment.

6. Select the *Activities* tab (top left) and drop the *MAIL\_RECEIVE* activity into the processing sequence. Locate *MAIL\_RECEIVE* in the *Email* group.

This must be within the *While* loop. You can adjust it to the correct level if necessary, using the  $\bigcirc$  Promote  $\bigcirc$   $\bigcirc$  Demote  $\bigcirc$  buttons.

- 7. Select the *Configurations* tab (top left)
  - a. Drop the iiiTUT\_POP3 Tutorial POP3 configuration into the Inbound POP3 Mail configuration ID. You defined this in Step 2.
  - b. Set the Subject of email to a value of &DEPARTMENT. The email subject line will contain a department code such as "ADM" which will be used to select employees from the file and output to the CSV file.
  - c. In the FROMADDRESS parameter, type &RCVFROM.

See below:



8. Save 💾 your processing sequence.

Save			×
	Type name and descr	iption for new processing sequence:	
	Name:	IIITUTSEQ06	
	Description:	iii Handle Employee List via Email - LIC008	
	Status:	Active	
	<u>Save</u>	Cancel	

Ignore the error and warning messages. You will complete the processing sequence definition in the following steps.

9. Once again select the Palette tab (top left) and drop an If directive after the EMAIL\_RECEIVE activity. Set the condition to &RCVFROM \*NE ''. (i.e. RCVFROM is not equal blank) RCVFROM is returned by MAIL\_RECEIVE. It will be blank if no email is found.

Your processing sequence should now look like the following:



#### 10. Select the *Transformations* tab (top left)

- a. Drop the iiiTUTMAP02 which you created earlier at the end of the processing sequence. Make sure it is positioned within the If loop and adjust if necessary using the promote/demote buttons.
- b. Select the *Configurations* tab (top left). Drop the *Database configuration* iiiTUTDEMO which you created earlier into the EARTHSourceConnection parameter.
- c. Select the *Variables* tab (top left) and drop the DEPARTMENT variable into the Department\_InSourceParameter. We used this variable for the Subject line on the MAIL\_RECEIVE activity.
- d. Set the TextfileTargetFilename to

/LANSA\_Composer\_licpgmlib/lic/Tutorial/\_SentEmail/iii\_Employ Your processing sequence should now look like the following:

📔 Details 🛛 📲 Parameters 🔲 Notes							
Item			Details				
				*** Beginning of processing :	sequence ***		
Θ	9	Assign					
	E	Expression	n:	&FLAG_WHILE = 'OK'		=	
	5	While				-	
	(	Condition:		&FLAG_WHILE = 'OK'			
E		Activ	ity				
		Name		MAIL_RECEIVE - Get single	email from server		
E	9	(+) If					
		Condit	tion:	&RCVFROM *NE			
		🗆 🌀 T	ranst				
		Na	ame:	JMITUTMAP02 - Employee F	ile to CSV with Select	tion 👻	
Deta	ails	Parame	ters				
			Variable	or value for parameter	Parameter Name	Parameter description	
1	1 * Inbound 'JMITUTDBID' JMITUTMAP02_1 D4CLICLIBSource			D4CLICLIBSourceConnection			
2	2 * Inbound &DEPARTMENT			RTMENT	JMITUTMAP02_2	Department_InSourceParame	
3	3 * Inbound '/LANSA_Composer_D4CPGMLIB/lii JMITUTMAP02_3 TextfileTargetFilename					TextfileTargetFilename	

- 11. Save your processing sequence.
- 12. Select the *Palette* tab (top left) and drop an *Assign* directive at the end of the processing sequence. This will be used to set a value to a variable TONAME which will be merged into the response email text. Complete the expression:

&TONAME = 'Personnel Administrator'

Your processing sequence should now look like the following:

📔 Details 🛛 👆 Parameters 🛛 🛄 N	lotes				
Item	Details				
	*** Beginning of processing sequence ***				
🖃 🛅 Assign					
Expression:	&FLAG_WHILE = 'OK'				
🖃 👏 While					
Condition:	&FLAG_WHILE = 'OK'				
🖃 🍰 Activity					
Name:	MAIL_RECEIVE - Get single email from server				
□ ( <sup>4</sup> ) If					
Condition:	&RCVFROM *NE ''				
🖃 🖾 Transform					
Name:	JMITUTMAP02 - Employee File to CSV with Selection				
🗆 📄 Assign					
Expression:	&TONAME = 'Personnel Administrator'				
	*** End of processing sequence ***				
-Details:					
Expression:					
&TONAME = 'Personnel Administrato	r -				
	and the second s				

13. Select the *Activities* tab (top left) and drop the TEXT\_SUBSTITUTE activity at the end of the processing sequence, as shown below:

	De	tails	🔊 Pa	rameters	🛄 Notes				
Item	n				Details				F
*** Beginning of processing sequence ***									
🖃 🎒 Assign									
	_	Express	ion:		&FLAG_WHILE = 'OK'				
	0	While							
1	_	Conditio	on:		&FLAG_WHILE = 'OK'				
1		📸 Ac	tivity			16			
1	_	Nar Za ze	ne:		MAIL_RECEIVE - Get single ema	all from server			
		(*) II Cor	dition		SD CVED OM #NE !!				
			Tran	sform	ORCOPROFIL INE				
			Name		1MITLITMAP02 - Employee File t	o CSV with Selection			
			Assi	on					
			Expre	ession:	&TONAME = 'Personnel Adminis	strator'			
		🗆 🍰	Activ	vity					
			Name	4	TEXT_SUBSTITUTE - Substitute	into text skeleton			
					*** End of processing sequence	***			
									-
_		0		ni i					
Det	tails	Para	neters						_
				Variable or	value for parameter	Parameter N	ame Pa	arameter description	
1	*	Inbo	und			TEXTSKELET	ON Pa	ath & file name of skeleton	
2	*	Inbo	und			TEXTFILE	Pa	ath & file name for expanded text	
3	3 Inbound " <i>*5KELETON</i> " TEXTCCSID CCSID for the output text file								
				1000		2 000200000 D			

a. Complete the Path & filename of skeleton as:

'/LANSA\_Composer\_licpgmlib/lic/Tutorial/iii\_EmployeeInfo.txt'

b. Complete the Path & filename for expanded text as:

#### '/LANSA\_Composer\_licpgmlib/lic/Tutorial/\_SentEmail/iii\_Employee

- 14.Select the *Activities* tab (top left) and drop the MAIL\_SEND activity at the end of the processing sequence.
  - a. Select the *Configurations* tab (top left) and drop your iii\_INQ\_EMP SMTP email details configuration into the *SMTP Mail details configuration ID* parameter
  - b. Drop the iiiSMTPID Tutorial SMTP configuration into the *SMTP mail server configuration* parameter.
  - c. Set the Body text path and filename to:

/LANSA\_Composer\_licpgmlib/lic/Tutorial/\_SentEmail/iii\_Emplo

d. Set the Path and filename for email attachments to:

/LANSA\_Composer\_licpgmlib/lic/Tutorial/\_SentEmail/iii\_Emplo

eIn the TOADDRESS parameter, type **&RCVFROM** 

f.In the SMTPSUBJECT parameter, type **Employee List Request** 

Your processing sequence should now look like the following:

1	Deta	ails 🚽 Pa	rameters 🚺 Notes		
Item			Details		
			*** Beginning of processing sequence ***		
🖃 📋 Assign					
	E	xpression:	&FLAG_WHILE = 'OK'		
	יי	While .			
	n d	onation:	AFLAG_WHILE = OK		
	2	Name:	MAIL_RECEIVE - Get single email from server		
E	- T	If			
		Condition	: 8RCVFROM *NE ''		
	6	🗉 🔥 Trar	sform		
		Name	: JMITUTMAP02 - Employee File to CSV with Select	on	
	6	🗉 📄 Assi	gn		
		Expre	ission: 8.1 UNAME = Personnei Administrator		
		Name	TEXT_SUBSTITUTE - Substitute into text skeletor		
	6	🗄 🍰 Acti	/ity		
		Name	MAIL_SEND - Send email by SMTP		
			*** End of processing sequence ***		
					<b>*</b>
Deta	ails	Parameter:			
			Variable or value for parameter	Parameter Name	Parameter description
1	*	Inbound	'JMI_INQ_EMP'	SMTPMESSAGEDETAI	SMTP mail details configuration ID
2		Inbound	'JMITUTSMTPID'	SMTPSERVERID	SMTP mail server configuration ID
3		Inbound	&RCVFROM	TOADDRESS	Override TO address on detail configuration
4		Inbound		CCADDRESS	Override CC address
5	5 Inbound BCCADDRESS Overr			Override BCC address	
6		Inbound		FROMADDRESS	Override FROM address
7 Inbound Fred Smith FROMDISPLAYNAME				Override FROM display name	
Inbound Employee List Request     SMTPSUBJECT     Override SMTP mail subject				Override SMTP mail subject	
9	9 Inbound /LANSA_Composer_D4CPGMLIB/lic/tutorial/_SentEmail/jmi_Employ MAILBODYTEXT Body text path and file name				
10		Inbound	/LANSA_Composer_D4CPGMLIB/lic/tutorial/_SentEmail/jmi_Emplo	MAILATTACHMENT	Path and file name for mail attachment
11		Inbound		MAILATTACHMENTZIF	File name of zipped attachment

- 15. Save your processing sequence.
- 16. Select the *Palette* tab (top left) and drop an **Else** processing directive into the processing sequence. Then drop a **Terminate** below the **Else**. Use the *Promote / Demote* buttons if necessary to position these correctly. Your Processing sequence should now look like the following:

Item	Details	=
	*** Beginning of processing sequence ***	*
🖃 📄 Assign		
Expression:	8FLAG_WHILE = 'OK'	
🖃 👏 While		
Condition:	&FLAG_WHILE = 'OK'	
🖃 🏤 Activity		
Name:	MAIL_RECEIVE - Get single email from server	
□ {+} If		
Condition:	&RCVFROM *NE ''	
🖃 🆾 Transform		
Name:	JMITUTMAP02 - Employee File to CSV with Selection	
🖃 🛐 Assign		
Expression:		
	TEYT SUBSTITUTE - Substitute into text skeleton	
🗆 🚔 Activity		
Name:	MATL SEND - Send email by SMTP	
□ +() Else		
i i i i i i i i i i i i i i i i i i i	xxx <b>F</b>	
	End or processing sequence	
		+

- 17. Use the Save 💾 button to save your processing sequence.
- 18. Review the *Version History* tab for the *Processing Sequence* iiiTUTSEQ06. This will contain the versions saved during this exercise. **Note:** you can *View* any of these versions. *Restore* would make the selected version the current version.

1	Processing sequence : Ver	sion history (IIIT	UTSE005-iii Employee to CSV - LICOO7)	
	Details 📔 ≽ Run commands 🛛	📃 Run history 🤇	Tersion history 🔝 Notes 🛛 😣 Audit	
	Date / Time	User	Internal Identifier	🔄 Refresh
	102/2009 9:52:09	JIVORY	A2E62F3E8B4D42AE9655BFDA678973BC	
	🕵 25/02/2009 9:51:23	JIVORY	47D8ED48BA414DB4A9A24B58A12AFFAF	S ⊻iew
	🚰 24/02/2009 17:29:30	JIVORY	E36C69CACB1443D6B52FE72134E6E103	
	🚰 24/02/2009 17:29:25	JIVORY	1A852677E89843DCB3918B16531E58FE	💥 Delete
	<b>6</b> 24/02/2009 17:18:50	JIVORY	BD9142C65B1D41ED96A3EAED8EF96D4D	
				Restore
	and the second s	and the second second	and the second second distances and the second s	Come of a come

- 19. Use your email account to send an email request. In the subject line, enter a department code (for example ADM, AUD or FLT). Send it to the second email account which you specified as the FROMADDRESS in theEMAIL\_RECEIVE activity.
- 20. Use the **Run** button to open the run dialogue and then click the *Run* button to run it. If your processing sequence is correct, your log will look like

#### the following:

-	Processing sequence	JMITUTSEQ06					
٠	Description	ii Handle Employee List via Ema	il - LICOO8				
	Run number	00000000000024					
	Start date / time	21/09/2012 17:16:56					
	End date / time	21/09/2012 17:17:02					
	Execution result code	OK 💌					
		MAIL_RECEIVE riting Get single email from server ingle POP3 mail message ended starting: Employee File to CSV with te transformation map ended tivity TEXT_SUBSTITUTE te transformation map ended starting: Substitute into text skeleto tivity MAIL_SEND viating: Get angle email from server ingle POP3 mail message ended noe ended	Selection	Messa Messa Messa	je ID: je type: je text:	DXS0110 Elapsed (seconds): 5,959	
	۲			,			

21. Check the inbox for your email account which you specified in your iii\_INQ\_EMP SMTP Email Configuration. You should have a response email with the file iii\_EmployeeInfo.csv attached. Open this in Excel and check that it contains the expected list of employees.



22. Send a number of email requests for different departments (for example, ADM, FLT, AUD) and run your processing sequence again. Check your

email account inbox, which should contain a response for each request, with a CSV file attachment containing employees for the requested department.

You have now completed this exercise.

#### Summary

You have created a new processing sequence which:

- Retrieves a single email request using a supplied POP3 Activity
- Uses the transformation map you created in an earlier module, which extracts records from a physical file into a CSV file
- Sent a formatted email response with the CSV file as an attachment using the supplied MAIL\_SEND activity.

# LIC009 - Handle multiple requests via an email attachment Objectives

- To use POP3 to handle multiple email requests
- To create an extract XML file from a physical file
- use selection criteria received as a CSV email attachment
- FTP a response to a remote server.

## Steps

To achieve these objectives you will complete the following:

- Step 1. Create FTP configuration
- Step 2. Create Server Folders
- Step 3. Create Request and Response Sample Files
- Step 4. Create Transformation Map
- Step 5. Create Processing Sequence
- Step 6. Execute and Test the Processing Sequence
- Step 7. Handling multiple messages and response xml files Optional
- Summary

# **Before You Begin**

- You must have access to a POP3 email account
- You must have access to an FTP server/folder for upload.

Before completing this tutorial, you should have completed all the preceding tutorials.

#### **Step 1. Create FTP configuration**

- 1. In the Navigator, select Configurations, FTP configuration
- 2. Use the New button on the toolbar to create a new *FTP configuration* with:

ID:	iiiTUTFTPOUT
Description:	iii FTP Outbound Configuration

3. Set other configuration details as follows:

<b>FTP Туре</b>	Outbound
Status	Active
Remote Host	Use your own FTP server host name or IP address if accessible via the web.*
Remote User	Use your own FTP user name.***
Remote Password	Use your own FTP password.***
Local Directory Path	/LANSA_Composer_licpgmlib/lic /Tutorial/iiiFTPOUT
Remote Directory Path	Use your own FTP server folder.*

4. *Save* your configuration. It will look like this:
| 🎨 FTP configuration : Details (JMITUTFTPOUT-Tutorial FTP Outbound Config) |   |  |  |
|---|---|--|--|
| Details 🔯 Notes 🐼 Audit   |   |  |  |
| ID  | JMITUTFTPOUT                                    |  |  |
| Description   | Tutorial FTP Outbound Config                    |  |  |
| FTP type  | Outbound  |  |  |
| Status  | Active  |  |  |
| Remote Host   | EARTH   |  |  |
| Remote User   | TRAINER 11                                      |  |  |
| Remote Password   | **********                                      |  |  |
| Secure FTP (SSH2)   | No  |  |  |
| FTP Secure Command Channel  | No  |  |  |
| FTP Secure data Channel   | No  |  |  |
| FTP Datalink - Active/Passive   | Passive   |  |  |
| FTP Binary transfer Y/N   | Yes   |  |  |
| Local Directory Path  | /lansa_composer_k12pgmli/lic/tutorial/JMIFTPOUT |  |  |
| Remote Directory Path   | /lansa_composer_k12pgmlib/lic/tutorial/JMIFTPIN |  |  |
| PUT selection   | PUT all files                                   |  |  |
| Local Archive Path (OK)   |   |  |  |
| Local Archive Path (ER)   |   |  |  |
|   |   |  |  |

## **Step 2. Create Server Folders**

In this step, you are going to create two folders on the server, to hold your incoming and outgoing data files.

1. From Windows Explorer, navigate through the network to LANSA Composer server share, tutorial folder. The path will be something like this:

\\<SRV400>\LICComposer\lic\Tutorial

Where:

<SRV400> is the IBM i server name.

LIC is the LANSA partition which Composer is using.

LICComposer is a file share which maps to an IBM i server path such as:

/Lansa\_composer\_licpgmlib

- 2. Inside the Tutorial folder, create the following folders:
- iiiFTPOUT
- iiiFTPIN
- iii\_ReceivedEmail you created this folder in LIC008.

## **Step 3. Create Request and Response Sample Files**

In this step, you will create three text files and save them in the Tutorial folder:  $\\SRV400>\LICComposer\lic\Tutorial$ 

1. Use Notepad to create the Request CSV sample file. This file will contain a list of these employee numbers:

EmployeeNumber A0193 A1003 A1012

2. Save the file using the *Save as type: All Files* option so that you can specify the CSV file extension. Give it a name such as **iiiTUTEMPNO.csv**.

**Note:** Although the file is a Comma Separated File (CSV), you will not see commas because there is only one data column.

3. In a similar way, create the Response XML sample. The contents of iiiTUTEMPDET.xml will be like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Employees SYSTEM "iiiTUTEMPDET.dtd" >
<Employees>
<Employee>
<EmployeeNumber>A9999</EmployeeNumber>
<LastName>Smith</LastName>
<FirstName>Fred</FirstName>
</Employee>
<Employee>
<EmployeeNumber>A7117</EmployeeNumber>
<LastName>Doe</LastName>
<FirstName>John</FirstName>
</Employee>
</Employee>
```

**Note:** change the **iii** in <!DOCTYPE Employees SYSTEM "iiiTUTEMPDET.dtd" > to your initials.

4 Save the file as iiiTUTEMPDET.xml. Once again ensure you use the *Save* 

*as type: All Files* option to ensure Notepad saves the file with the .xml extension.

5. Create the response DTD file. The DTD file is used to describe the tags of XML files:

<!ELEMENT Employees (Employee)>

<!ELEMENT Employee (EmployeeNumber,FirstName,LastName,Address1,Address2,Address3,PostCc

<!ELEMENT EmployeeNumber (#PCDATA)> <!ELEMENT LastName (#PCDATA)> <!ELEMENT FirstName (#PCDATA)> <!ELEMENT Address1 (#PCDATA)> <!ELEMENT Address2 (#PCDATA)> <!ELEMENT Address3 (#PCDATA)> <!ELEMENT PostCode (#PCDATA)> <!ELEMENT Department (#PCDATA)>

6. Save the file as **iiiTUTEMPDET.dtd** ensuring that the filename extension is preserved.

**Note:** This file's name must be the same name as specified in the DOCTYPE in the xml file just created.

## **Step 4. Create Transformation Map**

In this step you will create a transformation map which will do the following:

- Select records from the Employee file (PSLMST) based on the employee codes from the file iiiTUTEMPNO.csv. Your processing sequence will receive the CSV file as an email attachment
- Write the selected employee information to the file iiiTUTEMPDET.xml
- 1. In the Navigator, select *Definitions*, *Transformation maps*
- 2. Use the button on the toolbar to create a new *Transformation map* with the following details:

ID iiiTUTMAP03

Description Employee details XML from CSV selection list.

Map Type Unspecified Type

Restartable Yes

Status Active

- 3. *Save* the transformation map.
- 4. Click the button to open the *MapForce* mapping tool.

You may get a warning message stating that LANSA Composer supports a subset of *MapForce* mapping functionality.

Click *Continue* to proceed to *MapForce*. You will see an empty map.

5. To insert the Request CSV file, choose **Insert Text File** using the *Insert* menu. Then select Use simple processing for standard CSV ... and press the *Continue* button.

	😢 Altova Mapi	Force - [IIITUTMAP03.mfd]			
	: 🐻 Eile Edit	Insert Project Component Connection			
	i D 🚅 🖪 🕼	4 XML Schema/File			
	🛃 🔂 🖬 🗐 🗐 🗐	🛐 Database			
	Libraries	EDI			
	🔁 соге	I Text file			
	aggregate	Web service function			
	avg	Excel 2007 File			
Insert Text	Component	× * * * * * * * * * * * * * * * * * * *			
Altova MapFo Please choos	orce provides different me e below:	hods to process text files depending on their complexity.			
O Use FlexT	ext module to process an	v kind of text files with complex, nested structures			
	Design graphically n	ew structure based on a sample file			
Open	n existing FlexText configu	ration file to map it with other sources/targets			
⊙ Use simple processing for standard CSV (delimited) and/or FLF (fixed-length) files					
		Continue			

6. In the Text import / export window, click the Input file button. Use this dialog to navigate to select the file \\

<SRV400>\LICComposer\lic\Tutorial\iiiTUTEMPNO.csv which you created earlier.

ínput / Output File		
\\Earth\k12composer\ic\Tutorial\JMITUTEMPNO.csv	Input file	OK
	Output file	Cancel

- 7. Check First row contains field names checkbox.
- 8. In the Text enclosed in radio button group, select Not. Your selection will look like this.

😢 Component Setti	ings	×
Input / Output File	Input file Output file ths relative to MFD file	OK Cancel
Input / Output Enco	oding	
Encoding name:	Unicode UTF-8	
Byte order:	Little Endian	
CSV Settings Field delimiter Tab Sem	nicolon  Comma  Space  Custom  Text enclosed in Not  ' *	<ul> <li>● CSV</li> <li>● Fixed</li> </ul>
EmployeeNumber	r	
string		<u> </u>
A0090		
A2564		
		4
Append Field	sert Field Remove Field << >>	

9. Press OK. Mapforce will display the file iiiTUTEMPNO.csv visualization.

Altova MapForce - [JMITUTM]	AP03A.mf	d*]	
🗞 <u>F</u> ile <u>E</u> dit <u>I</u> nsert <u>P</u> roject	t <u>C</u> omp	onent	Connection Function Output Vie
i 🗅 🚅 🔒 🕼 🕼 🗠 🗠	X 🖻	<b>a</b> >	🗙   🏘 🆓 🗽   🔧 🛒 85LT 85LT= 80. 🛛
🛱 🖯 1 🕅 1 🗐 🔀 🎬	™: 🕹 健	7 1	a 🕐 🖹 🖬 👘 🖬 🖓 🗗 Ϋ
Libraries	<b>▼</b> ‡	×	
avg count	result	Î	JMITUTEMPNO      File: JMITUTEMPNO.csv      Source () Rows      File: Suppose Number
max min	result	=	
string-join	result		-
	roout.		

10.Insert the *Database* element using the *Insert Menu*.

🙁 Altova MapFo	orce -	[JMITUTMAP03A.mfd*]		
😼 <u>F</u> ile <u>E</u> dit	Inser	<u>P</u> roject <u>C</u> omponent	Connection Function Output View	
i D 🚅 🖬 🧯	61 )	XML Schema/File 🍓 🦓 🎆 🔧 🚽 👬 👪 1999 🕻		
81 A1 841 E	0ı [	Database		
Liburnian	EDIJ E	EDI		
Libraries	∎ı ]	Text File		
core	<b>R</b> 1	Web Service Function		
aggregat	<b>X1</b> E	Excel 2007+ File A 🖓 🗍 File: JMITUTEMPNO.csv (>		
avg count		XBRL Document		

- 11.Use the existing *ODBC connection* which you created in an earlier module.
  - a. Login using your workshop IBM i *User ID* and *Password*. You may be prompted for a number of logins.
  - b. Upon successful connection to the database, you will get a Select Tables, Views to Insert dialog.

🖙 Insert Database Objects 🛛 🛛 🔀
Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.
Source         Image: Add/Edit SELECT Statement         Add/Edit Table Relations (0)         Image: EARTH ODBC         Image: EARTH ODBC<
Show Preview >> Insert Cancel

b. Expand the list of libraries, schemas or collections using the drop down menu next to the small arrow. After selecting the library, schema or collection containing the required file (for example, LICLICLIB or LC\_DTA on a Windows server), locate the file PSLMST and select it using the check box.

Note: the library name should be the same one used in the database connection string used when creating the *Database* configuration previously.

jdbc:as400://<SRV400>/<LIBRARY>;naming=sql;errors=full;date format=iso;true;translate binary=true

🖙 Insert Database Objects	
Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.	
Source	
💎 🔽 💠 🛛 Add/Edit SELECT Statement 🛛 Add/Edit Table Relations (0)	
	•
III SECTAB	
SKLTAB	
	~
Show Preview >> Insert Cano	;el

12.Click *OK* to return to the main window. You will see the PSLMST table Visualization displayed.

	EARTH_K12
	🖓 🔺 🗔 🕅 K12LICLIB.PSLMST 🖓
	EMPNO D
	SURNAME C
	🖒 🗔 GIVENAME 🖒
	ADDRESS1 🔁
-	ADDRESS2
	ADDRESS3
	POSTCODE 🔁
	phonehme 🖒
	💠 📖 PHONEBUS 🖒
	STARTDTER S
	🔶 🔤 TERMDATER 🖒
	💠 📼 🖬 DEPTMENT 🔅
	🗘 🗔 SECTION 🖒
	🗘 🗔 SALARY 🖒
	> <b>□ @@UPID</b>  >

13.From the *Insert menu*, insert the *XML Schema/File*.

😂 Altova MapForce - [IIITUTMAP03.mfd*] 🛛 🕴 🍦				
: 📴 <u>F</u> ile <u>E</u> dit	Ins	Insert Project Component Connection Function		
i 🗅 🚅 🔒 🖪	€îı	XML Schema/File		
🛃 📑 🗐 🔤	٦	Database	00 00 E	
Libraries	EDIŢ	EDI		
and core	∎ı	Text file		
aggregate	₽	Web service function		
avg	XLSX 281	Excel 2007 File		
count	prag	Construction to an and an and and	= Emplo	

a. Use the Insert menu, XML Schema/File ... navigate to the . . . .\Tutorials folder and select schema file iiiTUTEMPDET.dtd file.

b.Use the Browse button to select the XML file iiiTUTEMPDET.xml

Your XML file structure will be displayed in the map.c . Rearrange your mapping screen to look like this.



The map functionality required will select Employee Details from the database only when the Employee Number is listed in the CSV file.

- 14. From the toolbar or *Insert* menu, insert an SQL-WHERE Condition.
- 15. Connect the PSLMST table to the *SQL-WHERE* table input. This will open *SQL-WHERE* properties window.



16. In *SQL-WHERE* properties window, type in the select criteria: EMPNO = :inEmpNo

Press OK

SQL-WHER	E Properties	X
To create a pa Sample: Name SELECT (fields)	rameter, enter a ':' followed by its name in the WHERE condition. = :Name AND Age >= :ValidAge	
EMPNO = :int		*
Parameters:		Ŧ
Parameter	Туре	
inEmpNo	(auto-detect by database)	
	ОК Са	ancel

17.Connect the EmployeeNumber output from CSV file to the input inEmpNo of *SQL-WHERE* condition.

18.Connect the result output of *SQL-WHERE* to the Employee input element of the XML file. Your map should look like the following:



19.Finally, connect the data fields from the database outputs to the corresponding XML file inputs. Your final map will look like this:

JMITUTEMPNO	VI> I> I> I> I= K12LICLIB.PSLMST
EARTH_K12   EARTH_K12   EMPNO  SURNAME  GIVENAME  ADDRESS1  ADDRESS3  ADDRESS3  POSTCODE  ADDRESS3  BOSTCODE  ADDREBUS  CIFEMDATER  CIF	Impo       resulti         Impo       Impo         Impo </td

20.*Save* the map and close MapForce.

21.On the *Details* tab, click Prepare to generate the transformation map code.

Click *Prepare* on the *Prepare Transformation Map* dialog.

Remember to look for an *IBM i Signon* which may be hidden by other windows.

**Tip:** Use Windows Task Manager to switch to the *IBM i Signon* dialog.

Wait for the completion message and *Close* the *Prepare Transformation Map* dialog.

😵 Pr	repare T	ransformation Map		23
00	R	Transformation map Description	JMITUTMAP03 Employee Details XML from CSV Selection list	
2		Preparing transformation map	<u>P</u> repare	
<b>* * *</b>	Check Preproc Genera Compi Catalog	folders and files cess transformation map definition te code for transformation map <b>le code for transformation m</b> jue information for transformation m	LANSA MapCompiler	
	Deploy Clear w	transformation map executable file orking files and folders	s to LANSA Composer server	

## **Step 5. Create Processing Sequence**

In this step you will create a processing sequence to do the following:

- Assign a variable = OK
- While variable = OK
  - Retrieve single email using MAIL\_RECEIVE
  - If FROMADDRESS is not equal to blank (an email was found)
    - Loop through email attachment file (list)
    - CONCAT path and filename create output path\filename
    - Transform from file PSLMST to XML, selecting EMPNO form email attachment file
      - FTP XML file to remote host

Else

- Variable = END ends While loop next time through
- Terminate
- 1. In the Navigator, select Definitions, Processing sequences
- 2. Use the Brew button on the toolbar to create a new *Processing Sequence*.
- 3. Select *Palette* tab from the left panel, then drag Assign and drop it on the *Details* panel on the right side.
- 4. In the Expression input field of the *Assign* operation, type in &FLAG\_WHILE = 'OK'

🚰 Untitled*	(Untitled)	Processing see	juence e	ditor			
<u>File E</u> dit <u>V</u> ier	w Insert <u>H</u> e	elp					
Save 💾	Run Þ	🕹 💥 🐰 🖞		Þ Promote  🔂	0	)emote 🔿	
🤒 Variables	🖲 Built-ins				ſ	📔 Details 🛛 👆 Parame	eters
🥵 Palette	🍰 Activities	left Transformation	is   😚 Coi	nfigurations		Item	Details
PB		PE	RE	PE			*** Beginning of processing sequence ***
						🖃 📄 Assign	
Loop	While	Until	Leave	Continue		Expression:	&FLAG_WHILE = 'OK'
							End of processing sequence
			പ	631			
<b>NATION</b>			{ <b>`</b> }	{} <b>`</b>			
Switch	Case	Otherwise	lf	Elself			
⇒ເາ	<u> </u>			-F3			
13		C)					
Else	Activity	Transform	Catch	Assign		-Details:	
					Шí	Euprossion	
	$\bigcirc$					Expression.	
	<b>6</b>					«FLAG_WHILE = UN	
Suspend	Terminate	Comment					

5. From the left panel *Palette* tab, drag the While operation and drop it at the end of the sequence.

In the Condition field, type &FLAG\_WHILE = 'OK'

📔 Details 🛛 👆 Parame	ters
Item	Details 🦂
	*** Beginning of processing sequence ***
🖃 📄 Assign	
Expression:	&FLAG_WHILE = 'OK'
🖃 👏 While	
Condition:	&FLAG_WHILE = 'OK' *** End of processing sequence ***

6. From the left panel *Activities* tab, drag the MAIL\_RECEIVE activity and drop it at the end of the sequence.

Use *Promote / Demote* buttons if necessary to position the activity inside the While loop.

ltem	Details
	*** Beginning of processing sequence ***
🖃 🛐 Assign	
Expression:	&FLAG_WHILE = 'OK'
🗉 👏 While	
Condition:	&FLAG_WHILE = 'OK'
🖃 🌧 Activity	
Name:	MAIL_RECEIVE - Get single mail from server
	*** End of processing sequence ***

7. Complete the POP3CONFIG parameters of the MAIL\_RECEIVE activity by dragging the iii\_TUTPOP3 from the *Configurations* tab on the left panel or you can simply type it in.

Det	ails	Parameters	5		
			Variable or value for parameter	Parameter N	Parameter description
1	*	Inbound	'III_TUTPOP3'	POP3CONFIG	POP3 Mail configuration ID
2		Inbound		SAVEDIRECTC	Override save Directory
3		Outbound		POP3COUNT	Number of email messages rec
4		Outbound		FROMADDRES	From address of email
- 5-	L	Qutbound_	many from the strange	SHBJEST -	Subject of end

8. From the *Palette* tab, drag and drop an If at the end of the sequence. Make sure it is inside the While loop.

📔 Details 🛛 👆 Parame	eters 📋 Notes
Item	Details
	*** Beginning of processing sequence ***
🖃 📄 Assign	
Expression:	&FLAG_WHILE = 'OK'
🗆 👏 While	
Condition:	&FLAG_WHILE = 'OK'
🖃 🍰 Activity	
Name:	MAIL_RECEIVE - Get single email from server
🖃 🔁 If	
Condition:	&FROMADDRESS *NE ''
Details:	
Condition:	
&FROMADDRESS *NE	11

9. From the *Variables* tab, drag *FROMADDRESS* and drop it in the IF Condition field, then complete the condition by typing \*NE " (two single quotes, no space)

Note: The *FROMADDRESS* will be blank when no e-mail is received. You

will use this condition to change the loop flag later on to end the loop.

When the *IF* condition is true, then you will loop through the e-mail attachments (in case more than one attachment is received) and send the attachment to the transformation map, and FTP out the result.

- 10. From the *Palette* tab, drag the LOOP directive and drop it at the end of the processing sequence. It must be inside the IF condition block.
- 11. Fill in the LOOP fields by dragging and dropping ATTACHMENTLIST from the Variables tab, and by typing the other fields.

📔 Details 🛛 👆 Para	meters 🔲 Notes		
Item	Details	Seq	
	*** Beginning of processing seque	ence ***	-
🖃 📄 Assign		1	
Expression:	&FLAG_WHILE = 'OK'		
🖹 🕘 While		2	
Condition:	&FLAG_WHILE = 'OK'		=
🖻 🚔 Activity		3	-
Name:	MAIL_RECEIVE - Get single email	from server	
⊡ -{+} If		4	
Condition:	&FROMADDRESS *NE "		
E 🕚 Loop		5	-
List:	&ATTACHMENTLIST		
Loop	&ATTACHMENT		
Inde	&ATLOOPINDEX		
-Details:			_
Listname	ATTACHMENTLIST		
List Humo	ATTACHMENTEDT		-1
Loop variable	ATTACHMENT		
Index variable	ATLOOPINDE×		
L			

12. Drag the CONCAT activity from the *Activities* tab and drop it at the end of the sequence. It must be inside the LOOP block. Use the *Promote/Demote* buttons to correct the level if necessary.

You will use this activity to build the file name of the Response XML file as follows:

STRING 1	File path	'/Lansa_Composer_licpgmlib/lic/Tutori
STRING 2	File Name	'/iiiEmployeeDetails'
STRING 3	Attachment Count	&ATLOOPINDEX

STRING 4	File Type	'.xml'
STRINGOUT	ResultString	&RESPONSEFILENAME

De	Details Parameters						
			Variable or value for parameter	Parameter Name	Parameter description		
1	*	Inbound	'/lansa_composer_k12pgmlib/lic/tutorial/jmiftpout'	STRINGIN1	First string		
_2	*	Inbound	'/jmiEmployeeDetails'	STRINGIN2	Second string		
_3		Inbound	&ATLOOPINDEX	STRINGIN3	Optional further string (specify contigu		
4		Inbound	'.xml'	STRINGIN4	Optional further string (specify contigu		
5		Inbound		STRINGIN5	Optional further string (specify contigu		
6		Inbound		STRINGIN6	Optional further string (specify contigu		
7		Inbound		STRINGIN7	Optional further string (specify contigu		
8		Inbound		STRINGIN8	Optional further string (specify contigu		
_9		Inbound		STRINGIN9	Optional further string (specify contigu		
10	1	Outbound	&RESPONSEFILENAME	STRINGOUT	Result string		
-							

- 13. From the left panel *Transformations* tab, drag iiiTUTMAP03 and drop it at the end of the sequence. It must be inside the LOOP block. Use Promote/Demote to correct the level if necessary.
- 14. Fill in the *Transform Inbound* parameters by drag and drop from the *Variables* tab or typing:

		Variable or value for parameter	Parameter Name	Parameter description
*	Inbound	&ATTACHMENT	JMITUTMAP03_1	JMITUTEMPNOSourceFilename
2 *	Inbound	'JMIDB_400'	JMITUTMAP03_2	EARTHSourceConnection
3 *	Inbound	&RESPONSEFILENAME	JMITUTMAP03_3	JMITUTEMPDETTargetFilename

15. After the transformation has run, the processing sequence will use FTP to send the output XML file.

Drag FTP\_OUTBOUND activity from the *Activities* tab on the left panel. Set FTPCONFIG parameter to iiiTUTFTPOUT by drag and drop from *Configurations* tab or by typing.

Your sequence will look like this:

🖹 Details 🚽 Parameters						
Item	Details					
	*** Beginning of processing	*** Beginning of processing sequence ***				
🖃 📄 Assign						
Expression:	&FLAG_WHILE = 'OK'					
🖃 👏 While						
Condition:	&FLAG_WHILE = 'OK'					
🖃 嬦 Activity						
Name:	MAIL_RECEIVE - Get sing	gle mail from serv	ver			
□ <del>(+)</del> If						
Condition:	&FROMADDRESS *NE '					
🖃 🕐 Loop						
List:	ATTACHMENTLIST					
Loop variable:	AT ACHMENT					
Index variable:	ATLOUPINDEX					
	CONCAT - Trim trailing blanks and concatenate					
	CONCATE HIII (railing bio	anks and conca	teriate			
Name:	IIITIITMAP03 - Employee	s Details XML fr	om CSV List			
E 🚔 Activitu	Information Employee					
Name:	FTP_OUTBOUND - FTP Put files on remote host					
	*** End of processing sequence ***					
Details Parameters						
Variable or valu	ue for parameter	Parameter N	Parameter description			
1 * Inbound 'IIITUTFTPOU	T'	FTPCONFIG	FTP Configuration ID			
2 Inbound		REMOTEDIREC	Override remote directory tar			
3 Inbound		FILELIST_FTP(	Alternative list of files			
_4_ Inbound		LOCALDIRECT	Override local directory for file			
A.A. American M.A.	Suma Marchan	- Anna	and the second second second			

- 16. From the *Palette* tab, drag Else and drop it at the end of the sequence. It must be at the same level as the IF block.
- 17. The ELSE block will be reached when there is no further e-mail to process. Here you will add the logic to end the While block.

From the *Palette* tab, drag Assign and drop it at the end of the sequence. It must be inside the else block.

Type the assignment expression: &FLAG\_WHILE = 'END'

This will end the loop in the next iteration when the While condition is evaluated.

18. From the *Palette* tab, drag Terminate and drop it at the end of the sequence. Use the *Promote* button to shift the Terminate to the top level.

Your completed sequence should look like the following:



19. Save your processing sequence as:

Name:	iiiTUTSEQ07
Description:	Handle employee list CSV request – LIC009.
Status:	Active

#### Step 6. Execute and Test the Processing Sequence

- 1. You will need one or more CSV request files. You can use the iiiTUTEMPNO.CSV file you created in Step 3, or create your own CSV files. The files can contain any number of EmployeeNumber entries, each EmployeeNumber must be on a new line.
- 2. Using any e-mail client software, or web based e-mail, create an e-mail and attach the CSV test files.
- 3. Send the e-mail to the POP3 mailbox you are using in the exercise
- 4. In the Navigator, select Definitions, Processing sequences
- 5. Click the *Search* button to refresh the instance list. Select the *Processing Sequence* you have just created iiiTUTSEQ07.
- 6. Click, the *Run* button to bring up the Run processing sequence dialog. Click the *Run* button.
- 7. Using your preferred FTP client software, logon to your FTP server and folder where you should have uploaded the response files. If you are using a local server, use Windows Explorer to view the folder.

You should have produced a number of xml files. These are the responses containing Employee details.

The file names will be iiiEmployeeDetails1.xml, iiiEmployeeDetails2.xml ...etc. The number of xml response files is equal to the number of attachments you sent in your e-mail.

8. Examine the contents of the response files received. They will contain employee details for each employee number you specified in each CSV request file.

# **Step 7. Handling multiple messages and response xml files -Optional**

The exercise is designed to handle receiving multiple e-mails and process them. However, when processing the second and subsequent messages, the response xml files would have the same name as the xml files created for first e-mail response.

Find a way to keep all generated files, by manipulating the file names being generated.

*Hint: study how the exercise is handling multiple attachment situations, and use a similar approach.* 

#### Summary

In this exercise you have:

- Created an FTP Configuration
- Created a POP3 Configuration
- Created DTD and XML file definitions
- Created a Transformation Map which:
- Retrieves multiple email messages
- Creates an XML extract from the Employee File for each the employee numbers listed in CSV files attached to the email
- Created a Processing Sequence which:
- Retrieves multiple email messages
- Runs a Transformation Map for each CSV attachment file to create an XML extract file
- FTPs the XML files created for each CSV attachment file.

## LIC010 - Calling a Processing Sequence (Optional)

*LANSA Composer 3* introduced the ability for a processing sequence to call a processing sequence. This feature enables reusable processing sequences to be developed, which can be called from a number of other processing sequences. To support this feature, a processing sequence may now have outbound parameters defined.

#### Objectives

- To split processing sequence iiiTUTSEQ07 developed in exercise LIC009 into two parts
- iiiTUTSEQ07 will now call processing sequence iiiTUTSEQ7A which will contain the steps to:
  - 1. Use CONCAT to produce output file path and name
  - 2. Run transformation map iiiTUTMAP03 to extract Employees to XML file based on input CSV file of employee numbers
  - 3. Use FTP to send output employee XML file to output folder.

**Note.** This is a simple example of calling a processing sequence from another processing sequence. It does not represent an ideal solution. For example, since multiple employee XML files may be output by each run of iiiTUTSEQ07, it would be more efficient to include only steps 1 and 2 in processing sequence iiiTUTSEQ7A. After calling iiiTUTSEQ7A, iiiTUTSEQ07 could use a DIRECTORY\_LIST and FTP all output files to the output folder.

#### Steps

To achieve these objectives you will complete the following:

- Step 1. Create Processing Sequence iiiTUTSEQ7A
- Step 2. Modify Processing Sequence iiiTUTSEQ07
- Step 3. Test Processing Sequence iiiTUTSEQ07
- Summary

## **Before You Begin**

You must have completed exercise LIC009 in order to complete this exercise.

#### Step 1. Create Processing Sequence iiiTUTSEQ7A

- 1. Create a new processing sequence. This will immediately open the processing sequence editor.
- 2. Open processing sequence iiiTUTSEQ07 in the editor. Use the editor's Cut and Paste function to copy the three steps shown, into your new processing sequence.

**Note:** *Composer 4* enables you to *Copy* and *Paste* or *Cut* and *Paste* a number of processing steps in a single operation. Hold down the shift key to select the three activities required.

**Hint:** Focus on **\*\*\***End of Processing Sequence**\*\*\*** in your target (new) processing sequence before using *Paste*. This will ensure that you are pasting at the end of the sequence.



Your new processing sequence should look like the following:

📔 Details 🛛 👆 Parameters	
Item	Details
	*** Beginning of processing sequence ***
🖃 🊔 Activity	
Name:	CONCAT - Trim trailing blanks and concatenate
🖃 🔥 Transform	
Name:	JMITUTMAP03 - Employee Details XML from CSV
🖃 🊔 Activity	
Name:	FTP_OUTBOUND - FTP Put files on remote host *** End of processing sequence ***

- 3. Select the *Parameters* tab. Your new processing sequence requires two input parameters:
  - The **file name** to be processed by the transformation map.
  - The **loop index** to combine with the output filename in the CONCAT activity.
- 4. Select the *Variables* tab and drag and drop ATTACHMENT and ATLOOPINDEX on to the *Processing Sequence Parameters* tab.
- 5. Change the second column to *Required* for both parameters.

If you wanted to run this processing sequence stand alone you could also specify suitable parameter descriptions to display on the Run dialog.

6. Check that the *Parameters* are defined as **Inbound**.

			Parameter Name	Parameter description	Default v
1	Inbound	Required	ATTACHMENT		antoren harrier
2	Inbound	Required	ATLOOPINDEX		

7. *Save* your new processing sequence as iiiTUTSEQ7A – Extract Employees to XML & FTP

#### Step 2. Modify Processing Sequence iiiTUTSEQ07

- 1. Switch to processing sequence iiiTUTSEQ07, which should still be open in the editor.
- 2. Select the *Variables* tab and drag and drop the following variables onto the *Processing Sequence Parameters* tab
  - ATTACHMENT
  - ATLOOPINDEX
- 3. Check that both parameter are be defined as **Outbound**. If necessary select this column for one of the parameters and use the drop down provided to select **Outbound**.

			Parameter Name	Parameter description
1	Outbound	Optional ATTACHMENT		
2	Outbound 🗸	Optional	ATLOOPINDEX	

4. Select the *View / Refresh* menu option so that the new processing sequence, iiiTUTSEQ07A will be shown in this open processing sequence.4. In the editor for processing sequence iiiTUTSEQ07, select the *Processing sequences* tab, and drag and drop iiiTUTSEQ7A into the sequence inside the LOOP directive.

**Hint:** Focus on the *Else* which follows the LOOP before performing the drag and drop. This should ensure processing sequence is added in the correct place. If necessary use the *Promote* and *Demote* buttons to position the execute processing sequence correctly.

Your processing sequence should look like the following:



5. *Save* processing sequence iiiTUTSEQ07.

#### Step 3. Test Processing Sequence iiiTUTSEQ07

- 1. Send an email to the address accessed by your POP3 configuration. The email must have an attachment of at least one CSV file containing a list of employee numbers.
- 2. Run processing sequence iiiTUTSEQ07. It should call iiiTUTSEQ7A for each input file retrieved by the POP3 activity. Check your resulting output XML files containing a list of employee details. The results should be similar to those obtained in exercise LIC009.

#### Summary

- This exercise has shown how one processing sequence can be called from another.
- See the *LANSA Composer* guide for further information.
- See also the shipped *Transaction Document Processing* sequences such as TXDOC\_INBOUND.
- When the nested processing sequence runs, it has its own independent variable pool.
- Communication of variables is only possible through processing sequence parameters.
- Nested processing sequences run under the same job number as the containing processing sequence.
- Log output is shown in the log for the containing processing sequence.
- Restarting of failed processing sequences is fully supported.

## LIC011 - Email notification service

- This is a summary exercise; you should build your solution based on the notes which follow.
- Suggested steps and hints are provided but are not detailed.

## **Before You Begin**

- You must have access to an FTP server/folder for download
- You must have access to an SMTP server.
- Before completing this tutorial, you should have completed all the preceding tutorials.

## Scenario

You are the owner of a notification service. You send e-mail notifications to your mailing list subscribers to inform them of new store sales or special offers as soon as they are published by the stores. The stores will publish their sales information in xml files, and will place them in a special folder on their FTP servers designed for your access.

The xml file structure is defined by you, and same structure is used by all the stores. The xml file contains information such as store name, items on sale, each item name and price, and offer expiration date.

## Objectives

To pull the sales information from each store's FTP server, process each file, and make a list of items on sale in a simple text file or CSV format. Send an e-mail to the mailing list subscribers with a file containing the list of items on sale attached.

## Steps

In order to achieve these objectives you will need to complete the following:

- Step 1 Design the XML & DTD file structure
- Step 2 Create "local" and "remote" folders
- Step 3 Create the transformation map
- Step 4 Define each store as a trading partner.
- Step 5 Create an FTP configuration for each store
- Step 6 Define text response file for the email body
- Step 7 Create the processing sequence
- Step 8 Run processing sequence and test

## Guidance

- 1. Use the XML & DTD created in the previous exercise as a guide
- The XML file will contain details of products on offer for a single store
- Consider the following structure for the XML:

Offers Store Name Offer - Product Number - Product Description - Price - Expiry Date

- The XML file contains a number of lines for each Store name.
- The dtd file should define the structure of the XML file and its data elements. See earlier exercise for an example and apply this to your store offers XML file. The XML file should refer to the dtd file.
- 2. You will need to create these folders on the server:
- an "iiiEmailAttch" folder to store your output CSV file
- an "iiiStore\_Local" folder to hold the XML files retrieved via FTP
- a "remote" folder for each trading partner to hold the XML files which you will retrieve using FTP. For example folders named: iiiStore1\_Remote, iiiStore2\_Remote.
- 3. Define a *Transformation Map* to merge text with values from the XML Store Offer file into a CSV file. The CSV file design will be done in MapForce
  - The output text file can be created as a CSV file with a single column, where you can concatenate the information into human readable text strings.
  - Each line should be constructed by concatenating text elements and variable values from the input XML file to produce a readable text string. For example, each line of your text file could be similar to:

Store XYZ is offering ABC item on sale for \$999.99 for a limited time only, offer valid until DDD.

where XYZ, ABC and 999.99, DDD are extracted from XML file.

- Consider the following steps toe define the transformation map in *MapForce*:
  - Insert the XML Store Offer file
  - Insert a text file (CSV) with a single string column
  - Insert constants containing each piece of text required
  - Use the CONCAT string function to build each row made up of constants and values from the XML file

- Link the output from the CONCAT function to the "row" in the CSV file

- Save and prepare your map.
- 4. The stores will be defined as trading partners. Create new *Trading Partners*: iiiStore1, iiiStore2.

Review Administration and Housekeeping / Code Maintenance / Trading partner installation-defined properties. As shipped, a property "EXAMPLE – Example installation-defined property" is defined. New *Trading Partners* will have this property, See the *Properties* tab for each trading partner.

Put a unique value, such as **iiiSTO**, into each trading partner's Example property. Use this in your processing sequence to loop through all trading partners and select your trading partners only.

Do not create default linked folders for the new trading partners.

- 5. You will also need an *FTP configuration* for each store. Link the *FTP configurations* for each store's FTP server to a corresponding store (trading partner).
- 6. MAIL\_SEND: Assume the mailing list is handled by the e-mail service com[pany. You only need to send one email for each xml file received. Send each e-mail to a single address which represents the mailing list service.
- 7. Define a text file which will be the **body text** in your sent email
- 8. Create a processing sequence to do the following:
  - Loop through all trading partners
    - If trading partner Example property = "iiiSTO"
      - If trading partner FTPIN not equal blanks
    - Retrieve XML offer file via FTP (saving into Stores\_Local folder)
  - Directory List of iiiStores\_Local
    - Loop through directory list
    - Use CONCAT to build path/filename for CSV file to be output by transform map

- Run Transform Map

- Directory list of iiiEmailAttch folder (contain output CSV files from transform map)

- MAIL\_SEND using text file. Attach CSV files based on directory

list

- Terminate.
# Appendix A. Install or Upgrade LANSA Composer

#### Overview

LANSA Composer consists of server and client components:

• The server components provide the run-time support for executing integration solutions designed with LANSA Composer, as well as the client support for the client software and web support for the browser-based operational interface.

You must install or upgrade the server components on a supported server platform. You may wish to create multiple server installations, for example to support separate development and production environments.

• The client components connect to an installed server and provide the design tools and environment for designing and deploying LANSA Composer solutions.

The LANSA Composer client software must be installed on each PC to be used for designing LANSA Composer solutions.

# 1. Install on Server

# 2. Install on Client(s)



- Composer server, including:
- LANSA run-time
- LANSA for the Web
- LANSA Integrator

- Composer client, including:
- Altova MapForce
- Java Development Kit

## **LANSA Composer Requirements**

For information on the system requirements for LANSA Composer refer to these headings:

Server Requirements

**Client Requirements** 

Database Connectivity Requirements (Server and client)

Refer to the Supported Platforms and Versions document, for information on supported server and client platforms and software compatibility.

## **Server Requirements**

This release of the LANSA Composer server requires:

- An IBM i or Windows Server platform currently supported by LANSA
- For Windows Servers you will need versions of the following currently supported by LANSA:
  - Microsoft Internet Information Server (IIS)
  - The Microsoft MSXML 6.0 XML parser
- A supported database system to contain the LANSA Composer database. On IBM i servers, LANSA Composer will use the integrated DB2 for IBM i database. On Windows servers, you will need to have installed and configured one of the database management systems (DBMS) currently supported for LANSA software, for example, a supported version of MS SQL Server or Oracle. Refer to the LANSA Web site for details of current supported database systems.
- A Java Virtual Machine (JVM) currently supported by LANSA for use with LANSA Integrator. In order to run Transformation Maps created with Altova MapForce version 2013 or later, your JVM must be at level 1.6 or higher. Some LANSA Composer activities new in version 5.0, require that your JVM be at level 1.7 or higher. They are WAIT\_FILESREADY and WATCH\_DIRECTORY. You can still use LANSA Composer with the JVM at levels earlier than 1.7, but you will not be able to use these activities.

If you are installing LANSA Composer Server on a Windows Server, a 32bit JVM is required, even if your operating system may be a 64-bit operating system.

- The LANSA Composer IBM i or Windows server install will install a complete LANSA Composer server system that includes the LANSA runtime components at the necessary levels. The following LANSA software levels are used:
  - LANSA Version 13 SP1 run-time at EPC level 131100 with LANSA Integrator at EPC level 131300 or later.
- Licenses for your LANSA Server system as described in Complete the Installation

#### IMPORTANT NOTE:

If you are upgrading from LANSA Composer version 4.0, you must first request new LANSA Composer server licenses. For more information refer to Appendix B. License LANSA Composer.

In order to use LANSA Composer's browser-based Operations Console, you will also need software components and configuration that supports LANSA web (WAM) applications. You should refer to LANSA product documentation for complete information, but the key requirements include:

- LANSA for the Web support installed and correctly configured. The LANSA Composer server installation will usually install and configure the web support for you, but some further manual configuration may be necessary.
- For IBM i servers you will need versions of the following currently supported by LANSA:
  - IBM HTTP Server powered by APACHE for CGI

Additional server software is required in order to use the supplied chart views in the *Transaction Document Statistics* page of the *LANSA Composer Document Manager*. Refer to *Requirements for Chart Presentation in Transaction Document Statistics* in Work with Transaction Documents for details.

The Windows and IBM i server upgrade routines for this version of LANSA Composer are designed and tested to upgrade LANSA Composer version 4.0 to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported. You must upgrade first to version 4.0 and then to version 5.0.

# **Client Requirements**

This release of the LANSA Composer client software requires:

- A Windows client platform, currently supported by LANSA, that is capable of connecting to your LANSA Composer server
- Microsoft Internet Explorer at a level currently supported by LANSA.
- Altova MapForce 2014 release 2 (2014R2) is supplied with this version of LANSA Composer. The LANSA Composer client install will install this version if Altova MapForce is not already installed. Where Altova MapForce is already installed, it is strongly recommended that it be upgraded to at least the 2014R2 version supplied.
- A supported 32-bit Java development kit (JDK) Note that the LANSA Composer client install can install the JDK if required.

The Java development kit (JDK) must be at a level that is capable of compiling the Java code for Transformation Maps to the required Java version as determined by the JVM version used by LANSA Integrator on the LANSA Composer Server. In order to run Transformation Maps created with Altova MapForce version 2013 or later, the JVM must be at level 1.6 or higher on your LANSA Composer server:

• The Microsoft XML Parser, MSXML6. This is not installed with the LANSA Composer client software but can be downloaded from the Microsoft web-site if it is not already installed on your windows client machine.

Microsoft Silverlight is required to be installed on the client computer in order to use the supplied chart views in the *Transaction Document Statistics* page of the *LANSA Composer Document Manager*. Refer to *Requirements for Chart Presentation in Transaction Document Statistics* in Work with Transaction Documents for details.

An installation of LANSA Composer client will require approximately 250mb free disk space, plus approximately 500mb for Altova MapForce and 250mb for the Java development kit (JDK).

# **Database Connectivity Requirements (Server and client)**

If you will be using the mapping tool to map data to or from database tables, you will also need to ensure you have the necessary server (JDBC) and client (ADO, ODBC or JDBC) database components installed. Refer to Database Connectivity Components and Drivers for more information.

# LANSA Composer Server on IBM i

The latest information concerning this version of LANSA Composer is available on the LANSA web site. We recommend that you refer to this before proceeding with the installation or upgrade.

When you install the LANSA Composer server on your IBM i computer, you have the choice of:

- installing a new copy of LANSA Composer, or
- upgrading an existing installation of LANSA Composer.

For a new installation, the LANSA Composer installer will install a complete LANSA run-time, web and integrator environment.

The installation on the server includes these steps:

Before You Begin Checklist

Call the Installation Program

Install LANSA Composer

or

Upgrade an Existing LANSA Composer installation User Profiles Created by the Installation Creating Further User Profiles for Use with LANSA Composer Complete the Installation Subsystems and Jobs

# **Before You Begin Checklist**

Before you begin installing LANSA Composer, check the following:

- Does your server system meet the minimum hardware and software requirements specified in Server Requirements and in LANSA's current Supported Platforms and Versions?
- Do you have the correct version of the LANSA Composer Server for IBM i software media? (LANSA Composer Server for IBM i is supplied on a single DVD that also contains the LANSA Composer Windows server and Windows client software.)
- Do you know the device name of the IBM i optical drive (default is often OPT01) ?
- Do you have access to the QSECOFR user profile? (This is the recommended profile to be used when installing the LANSA Composer software.)
- Do you have the necessary licenses for LANSA Composer Server on IBM i?
- If you are upgrading an earlier version of LANSA Composer, have you (a) backed up your existing LANSA Composer installation, and (b) run the LANSA Composer Database Housekeeping task? Both of these steps are strongly recommended.

If you answer Yes to all these questions, continue with Call the Installation Program.

# **Call the Installation Program**

You must use the QSECOFR or QOTHPRDOWN profiles during the installation. The use of other profiles or the removal of the special authorities will cause security and integrity alterations to the installed system.

- 1. Sign on to the IBM i as either **QSECOFR** or **QOTHPRDOWN**. The QSECOFR profile is recommended. The QOTHPRDOWN profile is normally created by LANSA software installations and will exist only if you have previously installed other LANSA software.
- 2. If you are about to upgrade an existing installation of LANSA Composer from an earlier version, you should make sure that no jobs or users are using the LANSA Composer system. To do this, use an appropriately authorized user profile (QSECOFR is recommended) to check for system locks as follows:

WRKOBJLCK <licpgmlib>/DC@A01 \*DTAARAWRKOBJLCK <licdtalib>/DC@F02 \*FILE \*FIRST

If locks exist, ask the users to sign off or wait until the batch jobs have ended before continuing with the upgrade.

3. If you are about to upgrade an existing installation of LANSA Composer from an earlier version, then you should end the HTTP server, the listener and the JSM. If LANSA Composer is installed in the default libraries, then you could use commands like these to end the jobs before beginning the upgrade:

#### ENDTCPSVR SERVER(\*HTTP) HTTPSVR(<licpgmlib>)ENDSBS <licpgmlib> OPTION(\*IMMED)ENDSBS <licjsmlib> OPTION(\*IMMED)

- 4. Insert the LANSA Composer Server for IBM i software disk into the IBM i optical drive.
- 5. If you are about to upgrade an existing installation of LANSA Composer from an earlier version, then before issuing the LODRUN command, make sure you have not previously invoked LANSA Composer in the interactive job that you are about to use to initiate the upgrade. Doing so can leave locks on LANSA Composer that will prevent the upgrade from succeeding.

If in doubt, signoff the current session and sign on again (as either **QSECOFR** or **QOTHPRDOWN as above) immediately before issuing the LODRUN command.** 

6. From an OS/400 command line (CALL QCMD), enter the following command to restore the software from the CD-ROM:

# LODRUN DEV(\*OPT)

7. After a short delay the installation program will display an initial menu:



To install a fresh copy of LANSA Composer (into its own LANSA system), type option 1 and press Enter. For further details, refer to Install LANSA Composer.

To upgrade an existing installation of LANSA Composer, type option 2 and press Enter. For further details, refer to Upgrade an Existing LANSA Composer installation

## **Install LANSA Composer**

When you choose to install a new copy of LANSA Composer, the installation program prompts for details of the new installation. In most cases, you can accept the default values suggested. Otherwise, complete the details as described below.

관 <mark>]</mark> Session A - [24 x 80]		x
<u>File Edit View Communication Actions Window He</u>	lelp	
LANSA	A Composer Version 5.0	
	Install/opgrade	
Required Parameters		
Prefix for library names	IC	
IFS file folder	*DEFAULT	
Listener Port	04545 Web Instance Port 00080	
Integrator TCP Port	04560 Admin TCP Port 04561	
Host CCSID	<u>00037</u> Client CCSID <u>00819</u>	
Auxiliary Storage Pool	0000001 1-32, *ASPDEV	
ASP device	*ASP Name, *ASP	
Joh queue	OBATCH NAME, "NONE	
Job queue library	*I TRI	
sob queue marary		
F3=Exit F5=Refresh F12=Cancel		
M <u>A</u> A	08/	033
🚽 1902 - Session successfully started		10

#### **Prefix for library names**

Specifies the prefix that will be used to generate the library names that will be used for the install. For example, the default value will result in libraries being created with names including LICPGMLIB, LICDTALIB, LICJSMLIB, LICCOMLIB and LICLICLIB.

#### **IFS file folder**

Specifies the name of the folder in the IFS into which the IFS components of LANSA Composer will be installed. The installation program will create the folder if it does not exist. The special value \*DEFAULT will cause the installation program to generate a path that is specific to the LANSA system and partition. This is the recommended value.

#### **Listener Port**

Specifies the communications port number that will be used by the communications listener for this system. This is the communications port

through which LANSA Composer client installations will connect to this server.

#### Web Instance Port

Specifies the communications port number that will be used by web jobs to this system.

#### **Integrator TCP Port**

Specifies the communications port number that will be used by the LANSA Integrator Java Service Manager.

#### **Integrator Admin TCP Port**

Specifies the communications port number that will be used for the LANSA Integrator JSM console.

The default port numbers suggested will usually be acceptable when no other LANSA systems are present. In any event, but especially if you have existing LANSA or LANSA Composer systems installed, you should review and, if necessary, re-allocate the port numbers to avoid conflict with other LANSA or non-LANSA applications on your system. If in doubt, you should consult your network administrator.

## Host CCSID

Specifies the host CCSID used by the web server for data translation purposes.

#### **Client CCSID**

Specifies the client CCSID used by the web server for data translation purposes.

#### **Auxiliary Storage Pool**

Specifies whether LANSA Composer is installed to the system ASP (auxiliary storage pool number 1), to a basic user ASP (ASP numbers 2 through 32) or to the ASP device named in the next field (\*ASPDEV).

#### ASP device

If \*ASPDEV is specified for the *Auxiliary Storage Pool* field, this value specifies the name of the ASP device to which LANSA Composer is installed.

#### ASP group Name

If \*ASPDEV is specified for the *Auxiliary Storage Pool* field, this value specifies the name of the ASP group to which LANSA Composer is installed.

#### Job queue / Job queue library

Specifies the name and library of the job queue to which the LANSA Composer

installation will be submitted.

When you have reviewed and/or completed the necessary details, press Enter to proceed. A confirmation screen will be displayed – press F8 to confirm your choices and submit the installation job to batch.

WARNING: Do NOT attempt to use LANSA Composer in any way until the installation or upgrade has completed. Do NOT start the HTTP server, the listener or the JSM. Do not attempt to connect a LANSA Composer client to the LANSA Composer server. Do NOT attempt to use the COMPOSER or LANSA commands.

Refer to Complete the Installation for information on the next steps.

## **Upgrade an Existing LANSA Composer installation**

**IMPORTANT NOTE:** 

If you are upgrading from LANSA Composer version 4.0, you must first request new LANSA Composer server licenses. For more information refer to Appendix B. License LANSA Composer.

If you are upgrading LANSA Composer from an earlier version, you must upgrade first to version 4.0 and then to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported.

NOTE: As with any upgrade, you are strongly advised to make a backup of the LANSA Composer server installation before proceeding with the upgrade. You should also run the LANSA Composer Database Housekeeping task to cleanup, maintain and reorganize the LANSA Composer database.

The LANSA Composer installation program presents a list of LANSA Composer systems found on the IBM i computer. Type a 2 next to the system to which you wish to apply the upgrade.

<b>∋∏</b> Se	ession A - [24 x 80]									
<u>F</u> ile	<u>Edit View Commu</u>	unication <u>A</u> ctions	<u>W</u> indow <u>H</u> elp							
	LANSA Composer Version 5.0									
	Install/Opgrade									
Ту	pe options,	press En	ter.							
	2=Upgrade	LANSA COM	poser							
	LANSA	Version	Composer	Integrator	Web	IASP				
	System		Installed	Installed	Installed	Device				
	LCFPGMLIB	12	Yes	Yes	Yes	*SYSBAS				
	LCJPGMLIB	12	Yes	Yes	Yes	*SYSBAS				
	LCTPGMLIB	12 SP1	Yes	Yes	Yes	*SYSBAS				
	LC3PGMLIB	12	Yes	Yes	Yes	*SYSBAS				
	LC5PGMLIB	13 SP1	Yes	Yes	Yes	*SYSBAS				
	LICPGMLIB	13 SP1	Yes	Yes	Yes	*SYSBAS				
_	XFJPGMLIB	12 SP1	Yes	Yes	Yes	*SYSBAS				
	XF2PGMLIB	11 SP5	Yes	Yes	Yes	*SYSBAS				
-	XF3PGMLIB	11 SP5	Yes	Yes	Yes	*SYSBAS				
	XF4PGMLIB	12 SP1	Yes	Yes	Yes	*SYSBAS				
-						0100/10	More			
F	3=Exit F5=R	efresh F1	2=Cancel							
MA	A						11/002			
5	1902 - Session successfi	ully started					1			

The installation program prompts for the name of the partition in that system

that contains the installation of LANSA Composer that is to be upgraded. In a standard LANSA Composer installation, there will be only one partition named LIC. To proceed, complete the details as described below.

과] Session A - [24 x 80]									
<u>Eile Edit View Communication Actions Window H</u> elp									
LANSA Compose	LANSA Composer Version 5.0								
Instant	/ opgrade								
LANSA System: LICPGMLIB	Job queue: <u>QBATCH</u>	_							
Type options, press Enter.	Job queue library: <u>*LIBL</u>	_							
2=Upgrade LANSA composer									
LANCA Deutitien Deteile									
LANSA Partition Details									
<pre>_ LIC LANSA Composer</pre>	LICLICLIB								
		Bottom							
F3=Exit F5=Refresh F12=Cancel									
M <u>A</u> A		04/061							
1902 - Session successfully started		1.							

#### Job queue / Job queue library

Specifies the name and library of the job queue to which the LANSA Composer upgrade will be submitted.

#### **LANSA Partition Details**

In a standard LANSA Composer installation, there will be only one partition named LIC. Type a 2 next to the partition that you wish to upgrade and press Enter. A confirmation screen will be displayed – press F8 to confirm your choices and submit the upgrade job to batch.

WARNING: Do NOT attempt to use LANSA Composer in any way until the installation or upgrade has completed. Do NOT start the HTTP server, the listener or the JSM. Do not attempt to connect a LANSA Composer client to the LANSA Composer server. Do NOT attempt to use the COMPOSER or LANSA commands.

Refer to Complete the Installation for information on the next steps.

**Note:** If the LANSA system containing LANSA Composer is not at the level required by this version of LANSA Composer, the install

program will notify you.

For a standard LANSA Composer installation (where LANSA Composer was originally installed into its own new LANSA system), the install program will give you the option of upgrading the LANSA system during the LANSA Composer upgrade.

You can proceed with the LANSA Composer upgrade only if you respond YES to the LANSA upgrade.

However, you should only respond YES if you are very sure that doing so will not adversely affect other applications or your LANSA development activities. A LANSA system that was installed as part of a LANSA Composer installation should never be used for other activities. Providing the LANSA system is only used for LANSA Composer, it should be safe to proceed with the upgrade.

Earlier versions of LANSA Composer allowed the application to be installed into a pre-existing LANSA system. In such cases the install program will not allow the upgrade to proceed if the LANSA system is not at the level required by this version of LANSA Composer. You must quit the installation and upgrade the LANSA system to the required level before attempting the upgrade again.

# **Complete the Installation**

When the installation is complete a message will be sent to the user message queue for the user who submitted the installation. Check the messages and any output produced to verify that the installation was successful.

Once the installation is complete, you will need to apply any server licenses that you have been issued for LANSA Composer. For more information refer to License the LANSA Composer Server.

**Note:** The installation of LANSA Composer server is not complete until at least one client install has been completed and has successfully connected to the server. Loading of supplied activity and other definitions and setting initial system values for the correct operation of the system are performed from the client. Refer to Initialize Data and Settings for further information.

## **User Profiles Created by the Installation**

The successful installation of LANSA Composer on your IBM i server may have created new user profiles on your system:

• User QOTHPRDOWN

If this is the first LANSA software product to be installed on your system, then the installer will have created user QOTHPRDOWN, which by default is the owner for most of the objects associated with LANSA Composer (and other LANSA products);

• User LICPGMLIB (or similar name)

The installer created a user with name LICPGMLIB (or similar name if you used a library name prefix other than the default LIC). This user profile is configured for normal access to most LANSA Composer functionality and may be used to connect to your LANSA Composer server from the LANSA Composer client software.

In both cases, the profiles are created with default passwords. You may wish to assign new passwords and/or to review the user profiles to ensure that they conform to your security policies.

## **Creating Further User Profiles for Use with LANSA Composer**

If you wish not to use the supplied default user profile for your work with your LANSA Composer server, you may create your own profiles.

When creating a user that will be used to connect to your LANSA Composer server from the LANSA Composer client, you should create it with the job description that was created in the LANSA Composer program library (usually LICPGMLIB).

For example, you could use the following command to create LCUSER1:

CRTUSRPRF USRPRF(LCUSER1) TEXT('LANSA Composer user') JOBD(LICPGMLIB/LICJOBD)

Although this user profile has no special authorities, it should suffice for most LANSA Composer work in a default installation.

However, a lack of authorities may become evident in some particular respects. For example, this user may not be able to successfully prepare Transformation Maps due to a lack of specific authority to the /LANSA\_licpgmlib/jsm/instance folder in the IBM i IFS.

Most such authority-related errors can be avoided by making the user a member of the group LICPGMLIB (or similar name if you used a library name prefix other than the default LIC). For example, you could revise the command you use to create the user profile as follows:

CRTUSRPRF USRPRF(LCUSER1) TEXT('LANSA Composer user') JOBD(LICPGMLIB/LICJOBD) GRPPRF(LICPGMLIB)

If you have more than one installation of LANSA Composer on your IBM i server (for example, separate design and production environments), then you will usually need to use a different user profile to connect to each. In each case the user profile should specify the job description appropriate to the target LANSA Composer system.

# Subsystems and Jobs

The LANSA system in which LANSA Composer runs is a complete application execution environment supporting client/server as well as web applications. As such, it has a number of components that must be active on the server to provide the necessary application support.

When the installation is completed into a new LANSA system, the necessary subsystems and jobs are started for you. If you stop these subsystems or restart your system, you will need to restart the subsystems that provide LANSA application support for LANSA Composer.

## Starting the listener

The LANSA listener provides communications services necessary to support both the client/server and web parts of LANSA Composer. The LANSA listener must be active in order to start the LANSA Composer windows client or to open the LANSA Composer operations console in a web browser.

The LANSA listener is started by starting the subsystem in which the listener jobs run. By default this subsystem has the same name as the program library used for the LANSA installation. So if you installed LANSA Composer into a new LANSA system using the suggested defaults, you would start the listener with this command:

```
STRSBS SBSD(LICPGMLIB/LICPGMLIB)
```

# Starting the HTTP server

The HTTP server instance associated with the LANSA installation must be started in order to open the LANSA Composer operations console in a web browser.

The HTTP server instance is started by using the STRTCPSVR (Start TCP/IP server) command. By default, the HTTP server instance has the same name as the program library used for the LANSA installation. So if you installed LANSA Composer into a new LANSA system using the suggested defaults, you would start the listener with this command:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(LICPGMLIB)
```

# Starting the Java Service Manager (JSM)

The Java Service Manager associated with the LANSA installation must be

started in order to use LANSA Integrator services with LANSA Composer. This is necessary for many of the activities shipped with LANSA Composer.

The Java Service Manager is started by starting the subsystem into which it was installed. If you installed LANSA Composer into a new LANSA system using the suggested defaults, you would start the Java Service Manager with this command:

STRSBS SBSD(LICJSMLIB/LICJSMLIB)

## LANSA Composer Server on Windows

If you are upgrading LANSA Composer from version earlier than version 4.0, you must upgrade first to version 4.0 and then to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported.

You must first request new LANSA Composer server licenses before upgrading to version 5.0. For more information refer to Appendix B. License LANSA Composer.

A single copy of the LANSA Composer server can be installed on a Windows server system. If LANSA Composer server (version 5.0) is already installed, the installation program will allow you to repair it. Otherwise, it will allow you to install the LANSA Composer Server.

The installation/upgrade on the Windows server includes these steps:

#### For Upgrading from Version 4:

- Before You Begin Checklist
- Plan Your Upgrade from LANSA Create a Database for a New Composer Version 4.0
- Complete your Upgrade Plan using the **Upgrade Plan Notes**
- Immediately Before You Begin
- Start the Installation Program
- Complete the Installation

#### or you can

Repair an Existing LANSA Composer Installation

Any late breaking news about LANSA Composer is available on the LANSA web site. We recommend that you refer to this before proceeding with the installation or upgrade.

#### **For First Composer Installation:**

- Before You Begin Checklist
- LANSA Composer installation
- Start the Installation Program
- Complete the Installation

# **Before You Begin Checklist**

Before you begin installing or upgrading LANSA Composer, check the following:

- Does your server system meet the minimum hardware and software requirements specified in Server Requirements and in LANSA's current Supported Platforms and Versions.
- Do you have the correct version of the LANSA Composer Windows Server software media?
   LANSA Composer Windows Server is supplied on a single DVD that also contains the LANSA Composer Server for IBM i and the LANSA Composer Windows client software.
- Do you have access to a suitable (administrative) user profile on the Windows server?
- Do you have the necessary licenses for LANSA Composer Windows Server? If you are upgrading from LANSA Composer version 4.0, you must first request new LANSA Composer server licenses. For more information refer to Appendix B. License LANSA Composer.
- If you are upgrading an earlier version of LANSA Composer, have you (a) backed up your existing LANSA Composer installation, including the LANSA Composer database, and (b) run the LANSA Composer Database Housekeeping task? Both of these steps are strongly recommended.

If you answer Yes to all these questions, continue. If you are:

**upgrading**, go to Plan Your Upgrade from LANSA Composer Version 4.0. **installing**, go to Create a Database for a New LANSA Composer installation.

If you are upgrading LANSA Composer from an earlier version, you must upgrade first to version 4.0 and then to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported.

# Plan Your Upgrade from LANSA Composer Version 4.0

If you are upgrading LANSA Composer from an earlier version, you must upgrade first to version 4.0 and then to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported.

NOTE: As with any upgrade, you are strongly advised to make a backup of the LANSA Composer Windows server installation and the database that it uses before proceeding with the upgrade. You should also run the LANSA Composer Database Housekeeping task to cleanup, maintain and reorganize the LANSA Composer database.

# **Understand the Upgrade from LANSA Composer Version 4.0**

LANSA Composer Server Version 5.0 installs the Windows Server using the Windows Installer (MSI) technology recommended by Microsoft for Windows software installation and maintenance.

The use of this new installation technology will make future software installation and upgrades simpler and consistent with Microsoft recommendations.

However, for this one upgrade from version 4.0 to version 5.0, some special considerations apply. Principally, the new installer will not be aware of your existing LANSA Composer Version 4.0 installation.

Future upgrades from version 5.0 to later versions of LANSA Composer will recognize your version 5.0 LANSA Composer installation details.

#### **Overview of the Upgrade Procedure**

You will treat this as a new installation, including specifying a new and different installation path for the LANSA Composer Version 5.0 software. This will leave the version 4.0 software on your system, which will be removed afterwards.

For certain aspects of the new installation, you will need to provide the details that apply to your existing LANSA Composer version 4.0 installation:

- In particular, in the Setup the Local Database step you will need to provide the details that identify your existing LANSA Composer version 4.0 database. As a result, the version 4.0 database will be upgraded and become the database for the new LANSA Composer version 5.0 installation.
- Also in the Setup the Local Database step you must identify the path to the

existing LANSA Composer Server version 4.0 installation in the *Upgrade Folder* field. This is very important to permit the existing database to be successfully upgraded from version 4.0 to version 5.0 and to permit JAR files for transformation maps to be re-deployed to the new installation.

• Other details from your existing LANSA Composer version 4.0 installation that you may wish to re-use in the setup of LANSA Composer version 5.0 are covered in Complete your Upgrade Plan.

In addition, there are further considerations that may apply if you have other LANSA for the Web applications installed on your server. These considerations are described in item 7 of Complete your Upgrade Plan.

# **Complete your Upgrade Plan**

You will need some details from your existing version 4.0 installation to use when you are installing version 5.0 of LANSA Composer.

We recommend that you print the following table and complete it with the version 4.0 details that you will use.

Example entries are inserted in gray and numbers in brackets refer to guideline notes in Upgrade Plan Notes.

#### **Installation paths:**

New (version 5.0):	(1)	C:\Program Files\LANSA\LANSA Composer Server V5Plus
Existing (version 4.0):	(2)	C:\Program Files\LANSA Composer Server\x_win95\x_lansa\x_lic

#### Database:

DBMS Type:	(3)	MSSQL
Data source name:	(4)	COMPOSER
Server:	(5)	MYSERVER\SQLSERVER
Database name:	(6)	COMPOSER

#### Web:

Is the active IIS Plug-in installed in the

(7)

version 4.0 installation path?		<ul> <li> _  There is no active plug-in.</li> <li> _  No, another IIS plug-in is active.</li> <li> _  Yes, serving only LANSA Composer.</li> <li> _  Yes, serving other web applications too.</li> </ul>
JSM Virtual Folder name:	(8)	cgi-bin

#### **Communication Ports:**

Listener:	(9)	4545
JSM Server:	(10)	4560
JSM Administrator:	(11)	4561

# LANSA Composer Files Location:

Path: (relative to server)	(12)	C:\LANSA Composer
Network path: (relative to client)	(13)	\\MYSERVER\LANSA Composer
Path is child of installation path?	(14)	ü Select one  _  No  _  Yes - refer to notes.

#### **Customizations:**

Customizations:	(15)	Have you made custom additions or alterations to the files in the version 4.0 installation directories?
-----------------	------	---

#### Licenses:

New licenses	(16)	Have you requested and obtained new LANSA
required:		Composer server licenses?

# **Upgrade Plan Notes**

These notes refer to the reference numbers in the table in Complete your Upgrade Plan.

#### **Installation Paths Notes**

1. New version (5.0)

The new installation path must be different to the existing installation path for the version 4.0 LANSA Composer system. This is very important.

2. Existing version (4.0)

In the Setup the Local Database step you will need to specify the existing installation path down to the partition folder. That is to say, the path will usually end with  $x_win95\x_lansa\x_lic$ .

#### **Database Notes**

It is very important that you specify the database details during the installation that identify your LANSA Composer version 4.0 database. If you don't know these details, you can find them by opening the file *dxstart.cfg* in a text editor, such as notepad.exe. Look for lines similar to the following:

dbid=COMPOSER dbut=MSSQLS

Disregard any lines that begin with a semi-colon (;) as these are comments and are not effective.

3. DBMS Type

You can determine the existing version 4.0's DBMS (database management system) type by referring to the line beginning *dbut* = in the file *dxstart.cfg* (described above). It is very important that you specify the same DBMS type for the new version 5.0 installation.

4. Data Source Name

You can determine version 4.0's Data source name by referring to the line beginning dbid = in the file dxstart.cfg (described above). It is recommended that you use the same Data source name for the new version 5.0 installation.

5. Server

With the Data Source Name (from point 4), open the DSN in the *ODBC Data Source Administrator* program to find the name of the database server. It is very important that you specify the same database server for the new version 5.0 installation.

6. Database name

With the Data Source Name, open the DSN in the *ODBC Data Source Administrator* program (from point 5) to discover the database name. It is very important that you specify the same database name for the new version 5.0 installation.

#### Web Notes

7. Is the active IIS plug-in installed in the version 4.0 installation path? Depending on your response to this question, you may need to carry out further steps as described below.

First you must determine the location of your LANSA for the Web IIS plugin:

- a. Start the Internet Information Services (IIS) Manager.
- b. Open the ISAPI Filters feature.
- c. Locate the ISAPI Filter with the name 'lansaweb'.
- If there is no such ISAPI Filter, then select the option *There is no active plug-in* in this item in the Upgrade Plan. No further action is required.
- If there is an ISAPI filter, examine the executable file path associated with it. The path will usually end with something similar to: ...\WebServer\IISPlugin\lansaweb\lansaweb.dll.

If the path is a child of the LANSA Composer Version 4.0 installation path, then go to Is LANSA Composer the Only Web Application being served by the Active IIS Plug-in? to determine which of the *Yes* options to select.

If the path is not a child of the LANSA Composer Version 4.0 installation, select the option *No, another IIS plug-in is active* in your Upgrade Plan. No further action is required at this time, however you may need to follow the instructions in Configure IIS to Serve the LANSA Composer Web Components when you have completed the installation.

8. JSM Virtual Folder

You should use the same JSM Virtual Folder name that you used for the LANSA Composer version 4.0 installation. You can determine the existing system's JSM Virtual Folder name by opening the *Internet Information Services (IIS) Manager* and locating the folder in the web site that refers to the .\*Integrator\JSMCGI* folder in the existing version 4.0 LANSA Composer installation. The default name used by prior LANSA Composer installations

was cgi-bin.

#### **Communication Ports Notes**

It is recommended that you use the same port numbers that you used for the LANSA Composer version 4.0 installation.

9. Listener port number

If you don't know the listener port number used for the existing installation, you can discover it by opening the file *listen.dat* in a text editor, such as notepad.exe. The *listen.dat* file can be found in the *Connect* directory of the existing LANSA Composer Version 4.0 installation path.

#### 10. JSM Server port number

If you don't know the JSM port numbers used for the existing installation, you can discover them by opening the file *manager.properties* in a text editor, such as notepad.exe. The file *manager.properties* can be found in the *Integrator\JSMInstance\system* directory of the existing LANSA Composer Version 4.0 installation path.

Look for the *last* occurrence (there may be more than one) of a line beginning **tcp.port=**, and make a note of the port number as the JSM Server port number. If such a line is not present, the default is 4560.

#### 11. JSM Administrator port number

In the *manager.properties* file, described in item 10, look for the *last* occurrence (there may be more than one) of a line beginning **console.tcp.port=**, and make a note of the port number as the JSM Administrator port number. If such a line is not present, the default is 4561.

#### LANSA Composer Files Location Notes

12. Path (relative to the server)

To determine the path to the LANSA Composer Files location used in the existing version 4.0 installation, firstly determine the network path as described for item 13 below. Then translate that network path to the equivalent local path relative to the server on which LANSA Composer is installed.

13. Network path (relative to client)

To find the network path to the LANSA Composer Files location used in the existing version 4.0 installation, open the LANSA Composer version 4.0 client software, go to *System Settings*, and locate the *Server network path* setting.

14. Path is child of installation path?

If your existing LANSA Composer Files location is **NOT** a child directory of the LANSA Composer Version 4.0 installation path, then select the *No* option for this item. No further action is required.

If your existing LANSA Composer Files location is a child directory of the LANSA Composer Version 4.0 installation path, then select *Yes* for this item. For example, if the existing LANSA Composer version 4.0 is installed at: *C:\Program Files\LANSA Composer Server* and your existing LANSA Composer Files location is at *C:\Program Files\LANSA Composer Server* Server\DATA then your answer is *Yes*.

If *Yes*, then you have two options to consider:

- Proceed, using the existing LANSA Composer Files location. When you come to the post-installation task Remove LANSA Composer Version 4.0, you should make sure that you remove all the contents of the Version 4.0 installation directory **except** the directory that is the LANSA Composer Files location. You should **not** remove the entire Version 4.0 installation directory as this will remove the LANSA Composer Files location as well.
- Choose a **new** LANSA Composer File location for the Version 5.0 installation.

We recommend that you choose a location that is **not** in the *Program Files* (or equivalent) tree.

You may not wish to use a new location if your existing solutions contain hard-coded references to the LANSA Composer Files location or if other applications also use the same folders or files.

#### 15. Customizations

If you have made custom additions or alterations to the files in the version 4.0 installation directories, you should identify them so that you can re-apply the customizations to the new version 5.0 installation. Such changes might include:

• LANSA Communications Administrator entries that you may have created to permit your LANSA Composer solutions to communicate with other LANSA applications or other LANSA Composer installations, especially those on separate or remote server systems. Typically such entries would be associated with *LANSA system configurations* used in LANSA Composer solutions with the CALL\_FUNCTION or COMPOSER\_RUN activities. You can use the *LANSA Communications Administrator* program to identify any such entries.

- Custom LANSA functions that are called by your LANSA Composer solutions using the CALL\_FUNCTION activity. The affected files would usually consist of one or more DLL files in the partition-execute directory (typically .\X\_WIN95\X\_LANSA\X\_LIC\Execute). (Recommended practice for recent LANSA Composer solutions would be to call such functions in a separate LANSA environment. However, especially in earlier versions of LANSA Composer, you may have developed solutions that used functions deployed to the LANSA Composer installation directories.)
- The activity processors for custom activities. The affected files would usually consist of one or more DLL files in the partition-execute directory (typically .\X\_WIN95\X\_LANSA\X\_LIC\Execute).
- Custom LANSA Integrator (JSM) solutions used with LANSA Composer. Such solutions might include ones built using the LANSA Integrator Studio "binding" wizards (for SOAP server and agent or for XML or JSON bindings).

The affected files would usually consist of one or more JAR files in the *Bindings* or *JAR* directories of the LANSA Integrator directory tree (typically .\*Integrator\JSMInstance*). Such solutions may also necessitate new entries in properties files in the *Properties* directory.

• Changes to internal LANSA, web and LANSA Integrator (JSM) configuration files. Normally such changes would be made only in exceptional circumstances upon the advice of LANSA Support.

You do not need to identify software patches and hotfixes to version 4.0 that you obtained from LANSA Support as these are, where applicable, included in LANSA Composer version 5.0. Nor do you need to identify the JAR files for transformation maps as these will automatically be re-deployed to the new installation.

#### 16. Licenses

Before proceeding with your upgrade, you must first request new LANSA Composer server licenses. For more information refer to Appendix B. License LANSA Composer.

# Is LANSA Composer the Only Web Application being served by the Active IIS Plug-in?

To determine this:

- 1. Start the *Web Administrator* in the *Settings and Administration* folder of the LANSA Composer program group.
- 2. From the *Options* menu, choose *Local configuration*, and then *Open*.
- 3. In the *Open* dialog, navigate to the ..\*WebServer*\*IISPlugin*\ directory of your LANSA Composer version 4.0 installation, select the file *l4w3serv.cfg*, and click the *Open* button.
- 4. In the *LANSA Web Administrator* window, choose the *Maintain systems* command from the *Tools* menu.



• If there is only one entry:

Then select the option *Yes, serving only LANSA Composer* for item 7 in the Upgrade Plan.

• If there is more than one entry (ie: LANSA Composer is **NOT** the only web application being served by the active IIS plug-in): Then select the option *Yes, serving other web applications too* for item 7 in

the Upgrade Plan. After the Composer V5 installation, you must follow the steps in Restore Configurations for LANSA for the Web applications. This is to restore the IIS and LANSA for the Web configuration to a state so that it can continue to serve both LANSA Composer's web components and the other applications that are presently served by the active IIS plug-in.

Now close the LANSA Web Administrator and return to Complete your Upgrade

Plan.

# **Immediately Before You Begin**

When you have completed planning and are ready to start the installation, you should complete the following steps:

#### **Stop services**

Use the Microsoft Windows *Services* Management Console applet to stop and disable the JSM and listener services.

- The JSM service will have a name like *LANSA Integrator JSM Administrator*.
- The listener service will have a name like *LConnect Services*.

In both cases, if there is more than one similarly-named service, use the path contained in the *Description* column to identify the one that applies to your LANSA Composer Version 4.0 installation. When you have identified the correct services, stop them. Then display their properties and set the *Startup type* to *Disabled*.

#### De-activate the existing LANSA for the Web plug-in

If your selection in item 7 of your Upgrade Plan is *Yes*, (*either* of the *Yes* options), these steps are necessary:

- 1. Start the Internet Information Services (IIS) Manager, and stop the server.
- 2. In *Internet Information Services (IIS) Manager*, open the *ISAPI Filters* feature.
- 3. Locate the *ISAPI Filter* with the name 'lansaweb' and *Remove* it. If there is a filter with the name 'lansaweb64', then *Remove* it too.

When you have completed these steps, continue your upgrade with Start the Installation Program.

# Create a Database for a New LANSA Composer installation

**Note:** If you are upgrading or repairing an existing installation of LANSA Composer, the upgrade or repair will use the database that was specified when the installation was initially created. You do not need to create a new database, so you may skip this step.

On Windows servers, you will need to have installed and configured one of the database management systems currently supported for LANSA software, for example, a supported version of MS SQL Server, Oracle or Sybase. Refer to the Supported Platforms document on the LANSA web site for details of current supported database management systems.

When installing a fresh copy of LANSA Composer you will need to create a new, empty database to hold the LANSA Composer database tables, using the supported database management system of your choice. The LANSA Composer installation will create the necessary database tables, but you must create the database that will hold them. You should add the new LANSA Composer database to your backup procedures.

Suggested database name: COMPOSER, although you can use your own choice of database name.

COMPOSER is the database name used for the database set up example during the installation in the Setup the Local Database step.

🍢 Micr	rosoft SQL Server Management	Studio								
File	Edit View Tools Window C	Community Help								
Ne	ew Query   🛅   📸 📸   📑		1 -							
Object B	Explorer		- ∓ ×							
Connec	t• 🖳 🛃 🔳 🝸 🛃 📓									
- 6	FARM0101 (SQL Server 10.0.1600 -	SYD\user 1)		1						
	Databases									
1	System Databases									
	New Database					크니즈				
	Select a page	🔄 Script 🝷 📑 H	elp							
	General									
	Options	Database name:		COMPOSE	R2					
		0		, cdofoults						
		Owner:		Cuerauit >						
Ŧ		Use full-text in	Idexina							
÷										
Ŧ	Database files:									
		Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth				
		COMPOSER2	Rows	PRIMARY	3	By 1 MB, unrestricted growth				
		COMPOSER	Log	Not Applicable	1	By 10 percent, unrestricted growth				

**Note:** When creating a Sybase database (including SQLAnywhere 10 and 11), you must select the Ignore trailing blanks in string
comparisons option. This translates to "Blank Padding" = ON in the "Extended Information" page in the database properties.

**Note:** This database is for LANSA Composer's own database tables that hold the definitions of the business process integration solutions you create with LANSA Composer. It should *not* be in the same database as your application data. Your LANSA Composer solution can access your application database (for example in Transformation Maps) even if it is in another database, database system or even on another computer.

When you have created your database, continue with Start the Installation Program.

# Start the Installation Program

If you are upgrading from LANSA Composer version 4.0, then be sure you have completed the steps in Immediately Before You Begin.

Log on to your Windows server using a user that has administrative rights on the computer.

Insert the LANSA Composer installation DVD into the DVD drive of your Windows server. After a short pause, the LANSA Composer installation program should load. (If the installer does not start automatically, use Windows Explorer to locate and open the *setup.exe* file in the root directory of the installation DVD.)



Click the *Install or Upgrade LANSA Composer Windows Server Software* option to start the LANSA Composer Windows Server setup program.

Depending on whether LANSA Composer Server version 5.0 has already been installed on this computer, continue with one of the following:

Install or Upgrade LANSA Composer Windows Servers Software

Repair an Existing LANSA Composer Installation

### Install or Upgrade LANSA Composer Windows Servers Software

If LANSA Composer version 5.0 has not been previously installed on your Windows Server, you will be prompted to install it.

If you are upgrading LANSA Composer from version 4.0, you will use the same procedure as for a new installation, but for certain aspects of the new installation, you will need to provide the details that apply to your existing LANSA Composer version 4.0 installation. Please make sure you have read Plan Your Upgrade from LANSA Composer Version 4.0 and completed your *Upgrade Plan* before proceeding.

IMPORTANT NOTE:

If you are upgrading from LANSA Composer version 4.0, you must first request new LANSA Composer server licenses. For more information refer to Appendix B. License LANSA Composer.

If you are upgrading LANSA Composer from an earlier version, you must upgrade first to version 4.0 and then to version 5.0. Upgrading directly from earlier versions to version 5.0 is not supported.

NOTE: As with any upgrade, you are strongly advised to make a backup of the LANSA Composer Windows server installation and the database that it uses before proceeding with the upgrade.

The *LANSA Composer Server Setup* wizard will prompt for a number of options that are essential to the correct operation of your LANSA Composer system. It is strongly recommended that you read the descriptions of each step that follow, to ensure that you make the correct choices to ensure a successful first-time installation.

The *LANSA Composer Server Setup* is in two parts:

1. The *LANSA Composer Server Setup* wizard will install the LANSA Composer Server software on your system. The following topics describe the steps and the responses you should provide at each step:

Welcome to the LANSA Composer Server Setup Wizard Installation Scope Destination Folder Setup the Application Setup the Local Database User Id Web Sites for IIS Plug-In Web Site Virtual Folders Communication Ports Choose Setup Type Ready to install LANSA Composer Server Installing LANSA Composer Server Setup Wizard

2. The *Completing LANSA Composer Server Setup* wizard will then start and guide you through the further steps necessary to load LANSA Composer files and data in order to prepare your LANSA Composer installation for first use. The following topics describe the steps and the responses you should provide at each step:

Completing LANSA Composer Server Setup LANSA Composer Files Location LANSA Composer Files Network Path Import Supplied Definitions Ready To Install Installing LANSA Composer Files and Data LANSA Composer Files and Data Have Been Installed

# Welcome to the LANSA Composer Server Setup Wizard

If LANSA Composer version 5.0 has not been previously installed on your Windows Server, you will be prompted to install LANSA Composer Server.

If you are upgrading LANSA Composer from version 4.0, you will use the same procedure as for a new installation, but for certain aspects of the new installation, you will provide the details that apply to your existing LANSA Composer version 4.0 installation. These are the details you will have recorded in Complete your Upgrade Plan.

Please read all of the following section before you proceed:



Click *Next* and continue with:

### Installation Scope

If, instead, you see the following screen, it means that LANSA Composer version 5.0 is already installed on this Windows server.

Only a single copy of the LANSA Composer server can be installed on a Windows server system. If LANSA Composer server is already installed, the installation program will allow you to upgrade or repair it.



In this case, please continue with:

Repair an Existing LANSA Composer Installation

### **Installation Scope**

You will be prompted to choose the installation scope. This being a server application, you should always choose *Install for all users of this machine*.



Make sure that *Install for all users of this machine* is selected, then click *Next* and continue with:

**Destination Folder** 

# **Destination Folder**

You can specify the folder that LANSA Composer Server will be installed into. The *LANSA Composer Server Setup* wizard will create the folder if necessary.

For a new installation, we suggest accepting the default folder proposed.

If you are upgrading from LANSA Composer version 4.0, then you must specify a new and different installation path for the LANSA Composer Version 5.0 software. Refer to item 1 in Complete your Upgrade Plan.

NOTE: The destination folder must NOT be the existing LANSA Composer version 4.0 installation (ie: item 2 in Complete your Upgrade Plan). The upgrade will not be successful if it is.

Destination Folder	
Click Next to install to the d	lefault folder or dick Change to choose another.
Install LANSA Composer Serv	ver to:
C: Program Files \LANSA \LA	NSA Composer Server)
1-1-2	VOA Composer Server (
Change	
Change	
Change	

NOTE: If you are installing LANSA Composer Server on a 64-bit version of Windows, the suggested installation path will begin with something like C:\ProgramFiles (x86)\.... In this event, you must not attempt to change the installation path to C:\Program Files\.... Microsoft Windows reserves the latter path for 64-bit applications. LANSA Composer Server is a 32-bit application. If you attempt to install it under C:\Program Files\..., the installation will be corrupted

and will not execute correctly.

When you have made your choice, click Next to continue.

You may see a window that tells you that the IIS plugin for LANSA for the Web support is already installed. This may happen if you already have another LANSA application on your server that uses LANSA for the Web.

In order to ensure that this installation does not interfere with the continued normal operation of the existing application, the *LANSA Composer Server Setup* wizard will not configure the IIS plugin to serve LANSA Composer's web components. In this event, you will need to manually configure the web support after the installation is complete. Instructions for this are provided in the following topic:

Configure IIS to Serve the LANSA Composer Web Components



Now continue with: Setup the Application

# **Setup the Application**

The *LANSA Composer Server Setup* wizard shows a window that lets you choose the *DBMS Type* (database management system type).

B LANSA Composer	Server Setup	
Setup the Applic	ation the default application settings or edit as requir	ed 🚺
DBMS Type	MSSQLS	•
	<u>Back</u>	xt OK

For a new installation, you should choose the option that applies to the database you created in Create a Database for a New LANSA Composer installation. For example, if you created your database in Microsoft SQL Server, then you should choose *MSSQLS*.

If you are upgrading from LANSA Composer version 4.0, it is very important that you specify the same DBMS type for the new version 5.0 installation as is used for the existing LANSA Composer Version 4.0 installation. Refer to item 3 in Complete your Upgrade Plan.

When you have made your choice, click *Next* and continue with:

Setup the Local Database

### Setup the Local Database

The *LANSA Composer Server Setup* wizard shows a selection of settings for the LANSA Composer Server database.

If you are installing a new copy of LANSA Composer Server, the choices in this step relate to the database that you created in Create a Database for a New LANSA Composer installation.

If you are upgrading from LANSA Composer version 4.0, the choices in this step must relate to the existing database in which LANSA Composer version 4.0 is already installed. Refer to items 2, 4, 5 and 6 in Complete your Upgrade Plan.

Click next to install the	e default local database settings or edit as required
Data Source Name	COMPOSER
Gerver	MYSERVER\SQLEXPRESS
Database	COMPOSER
Jser	
Password	
ocal Database	V Setup Database 🔲 Upgrade V12 Database
Jpgrade Folder	C:\X_WIN95\X_LANSA\X_LIC\
	Browse

Complete your choices in accordance with the following:

*Data Source Name*: Specifies the data source name (DSN) that LANSA Composer will use to access the database. The *LANSA Composer Server Setup* wizard will create the DSN when you proceed with the installation.

The data source name must be unique amongst DSN names on this Windows server, must not exceed 32 characters in length and may not include any blank characters. (If you wish to review existing DSNs that already exist on this computer, you can do so using the *ODBC Data Source Administrator* program in Windows.)

*Server*: This is the name that the database server is called on the network. Note, the Server name is not the PC name, or, at least, not just the PC name. As an example, for SQL Server the default name for a database server is in the form "PC Name\Instance Name" - for example, "MY\_PC\SQLSERVER".

*Database*: Specifies the name of the database. (This does not apply for an Oracle database.) If you are installing a new copy of LANSA Composer Server, this should specify the name of the database that you created in Create a Database for a New LANSA Composer installation. If you are upgrading from LANSA Composer version 4.0, this must specify the name of the existing database in which LANSA Composer version 4.0 is already installed. (refer to item 6 in Complete your Upgrade Plan).

*User* and *Password*: Specifies the user name and password that you wish LANSA Composer Server to use to access the database. You can leave these values blank for a Microsoft SQL Server database in order to use trusted connections.

*Setup Database*: When checked, this option will install or upgrade the database tables and any shipped data into the nominated database. This is the required and necessary behavior for the *LANSA Composer Server Setup* and the checkbox is checked by default. You must leave it checked for a successful LANSA Composer Server installation, whether it is a new installation or you are upgrading from LANSA Composer version 4.0.

IMPORTANT NOTE:

The *Setup Database* checkbox must be checked for a successful LANSA Composer Server installation (including upgrading LANSA Composer version 4.0).

*Upgrade V12 Database*: For a new installation, this option does not apply. Do not check this checkbox for a new installation.

If you are upgrading from LANSA Composer version 4.0, then you must check this checkbox.

**IMPORTANT NOTE:** 

You must check the *Upgrade V12 Database* checkbox if, and only if, you are upgrading from LANSA Composer version 4.0.

*Upgrade Folder*: For a new installation, this option does not apply and it will remain grayed (disabled) because you have not checked the *Upgrade V12* 

Database checkbox.

If you are upgrading from LANSA Composer version 4.0, then you must specify here the path to the LIC partition folder in the existing LANSA Composer version 4.0 installation. The path you specify should include all the folders down to the partition folder level. That is to say, the path will should end with x\_win95\x\_lansa\x\_lic. Refer to item 2 in Complete your Upgrade Plan.

When you have made your choices, click *Next* and continue with:

User Id

# User Id

The *LANSA Composer Server Setup* wizard prompts for details of the User Id that is to be used to connect the web server to the LANSA Composer application.

ction	Use Existing User 👻	
Jser Id	PCXUSER	
assword	•••••	
onfirm Password		

A Windows User is required that can be used to log on to the LANSA Composer environment to execute web requests associated with LANSA Composer's *Operations Console*.

The User can be setup elsewhere or through this dialog.

• From the dropdown list, select whether to *Use an existing User*, *Create a New Local User* or *Not to Configure any User*.

If you select *Create New Local User*, the User will be created as a local user. If using an existing User, the user details will be verified to ensure they are correct. An existing User may be a local or a domain User.

- Enter the User Id to be used when using LANSA for the Web. You may enter an existing or new User Id. The User will be automatically added to the LANSA Users Group.
- Enter the password for the User Id.

 If creating a new user, re-enter the Password to confirm the new Password. You do not have to confirm the password if you are entering an existing user.
 When you have completed your choices, click *Next* and continue with: Web Sites for IIS Plug-In

### Web Sites for IIS Plug-In

The *LANSA Composer Server Setup* wizard asks you to choose the web-site(s) through which you wish to enable LANSA Composer's web features. The affected web features include:

- LANSA Composer's *Operations Console* web application;
- The LANSA Integrator JSM Direct support that is used to direct inbound HTTP requests through LANSA Composer.

Websites in the right hand	l list are the selected ones	s. Select a website in the left list and
dick "Add" to include it in f	he selected list. Or select	a website in the right list and dick
Web site 2 Web site 3	Add >>	Default Web Site

The web-sites shown are those that are configured in the Microsoft Internet Information Server (IIS). There will only be web-sites listed on the left if you have web-sites in addition to the default configured in IIS. In any event, usually you will choose just one web-site.

(If you choose more than one web site, then further advanced configuration will usually be required through the *Web Administrator* application in order to direct selected requests in the required way.)

When you have completed your choices, click *Next* and continue with:

Web Site Virtual Folders

## Web Site Virtual Folders

The *LANSA Composer Server Setup* wizard asks you to choose the web-site in which you wish to define the virtual folders used with LANSA Composer's web features.

/ebsite	Default Web Site 🔹
M Virtual Folder	
) Create new Virtual Folder	older
choose existing virtual i t	
ANSAXF4CGI C402GA2CGI ANSAXF3CGI C402DEMCGI ANSAXF5CGI ansaweb	Mapped Location: C:\Program Files\LC511B07 Server\Integrator\JSMCGI

A virtual folder (in this context) is a means of instructing Microsoft Internet Information Services (IIS) to re-direct a component of the URL used for an incoming web request (the "virtual" folder) to a physical folder location, possibly outside the Microsoft IIS folder tree.

In particular, the *LANSA Composer Server Setup* defines two virtual folders:

 The *Images* folder identifies the location of web images used in LANSA Composer's *Operations Console* web application. It is always called *Images* – you have no control over the virtual folder name. The virtual folder will be configured to re-direct image requests to the appropriate folder in the LANSA Composer Server installed location, for example, to:

C:\Program Files\LANSA\LANSA Composer Server\WebServer\Images

• The *JSM* folder identifies the location of the LANSA Integrator JSM Direct support that is used to direct inbound HTTP requests through LANSA Composer. The *LANSA Composer Server Setup* permits you to choose the virtual folder name. If you create a new virtual folder, it will be mapped to

the directory where the JSM Direct support is installed, for example, to: C:\Program Files\LANSA\LANSA Composer Server\Integrator\JSMCGI

In this step of the *LANSA Composer Server Setup* wizard, you can control the creation of the necessary virtual folders in two respects, as follows:

### Website

Of the websites chosen in the previous step, this specifies which is to contain the *Images* and *JSM* virtual folders.

Usually you will have chosen not more than one website in the previous step - its name will be shown and selected here and you need make no further change.

### JSM Virtual Folder

This allows you /to specify the name of a new virtual folder (default cgi-bin) or to choose an existing virtual folder that is used with the LANSA Integrator JSM Direct support.

In both cases the related LANSA Integrator files (jsmdirect.exe, jsmproxy.exe, jsmadmin.exe and their associated data, message and configuration files) will be installed in the JSMCGI folder under the LANSA Composer root directory.

If you select an existing virtual folder, the related Integrator files (jsmdirect.exe, jsmproxy.exe, jsmadmin.exe and their associated data, message and configuration files) will then be copied into the physical folder the chosen virtual folder is mapped to.

If you are upgrading from LANSA Composer version 4.0, then you should use the same JSM virtual Folder name as you used for the existing LANSA Composer version 4.0 installation. Refer to item 8 in Complete your Upgrade Plan.

When you have completed your choices, click *Next* and continue with:

**Communication Ports** 

## **Communication Ports**

The *LANSA Composer Server Setup* wizard prompts you to specify the TCP/IP communication port numbers that will be used by the services supporting the LANSA Composer installation.

🛃 LANSA Composer Ser	ver Setup				x
Communication Po Specify the TCP/IP po	r <b>ts</b> rt numbers for e	ach of the runr	ing services		
Listener	4545				
JSM Server	4560				
JSM Administrator	4561				
		Back	Next	Cance	

For more information on the services, refer to:

#### Services Used by the LANSA Composer Windows Server

Port numbers may be assigned to the following services:

*Listener*: Specifies the TCP/IP communications port number that will be used by the listener for this system. The listener is the means by which LANSA Composer client installations and LANSA Composer web components will connect to this server.

*JSM Server* : Specifies the TCP/IP communications port number that will be used by the LANSA Integrator Java Service Manager (JSM). The JSM provides the implementation that executes many of the LANSA Composer activities.

*JSM Administrator* : Specifies the TCP/IP communications port number that will be used for the LANSA Integrator JSM console. The LANSA Integrator JSM console can be displayed from within the LANSA Composer client

software and is used to monitor the functioning of the LANSA Integrator Java Service Manager.

If you are upgrading from LANSA Composer version 4.0, then you should use the same port numbers as were specified for the existing LANSA Composer version 4.0 installation. Refer to items 9, 10 and 11 in Complete your Upgrade Plan.

For a new installation, the default port numbers suggested will usually be acceptable when no other LANSA systems are present. In any event, but especially if you have existing LANSA or LANSA Composer systems installed, you should review and, if necessary, reallocate the port numbers to avoid conflict with other LANSA or non-LANSA applications on your system. You should also avoid using port numbers in the well-known port number range. If in doubt, you should consult your network administrator.

When you have made your choices, click *Next* and continue with:

Choose Setup Type

# **Choose Setup Type**

The *LANSA Composer Server Setup* wizard asks you to choose the setup type that best suits your needs.

Choose the setup type that best suits your needs  Typical Installs the most common program features. Recommended for most users.  Custom Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.  Complete	Choose Setup Type	
Typical         Installs the most common program features. Recommended for most users.         Custom         Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.         Complete	Choose the setup type that be	st suits your needs
Installs the most common program features. Recommended for most users.         Custom         Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.         Complete	Typical	
Custom Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users. Complete	Installs the most com	non program features. Recommended for most users.
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.	Custom	
Complete	Allows users to choos they will be installed.	e which program features will be installed and where Recommended for advanced users.
	Complete	
All program features will be installed. Requires the most disk space.	All program features v	will be installed. Requires the most disk space.
		Back Next Can
Back Next Car		

There are no optional components associated with the LANSA Composer Server installation. You should always choose the *Complete* option in this step.

Click the *Complete* button, and continue with:

Ready to install LANSA Composer Server

## **Ready to install LANSA Composer Server**

The *LANSA Composer Server Setup* wizard shows a summary of the main options chosen before proceeding with the installation. If necessary, you may click *Back* to return to any of the previous steps to change your choices.



When you are satisfied that the choices are correct, click *Install* to begin the installation and continue with:

Installing LANSA Composer Server

### **Installing LANSA Composer Server**

As the installation proceeds, the *LANSA Composer Server Setup* wizard shows the progress in the *Installing LANSA Composer Server* window.

ANSA Composer Server Setup	
Installing LANSA Composer Server	
Please wait while the Setup Wizard installs LANS/	A Composer Server.
Status:	

### IMPORTANT NOTE:

Some of the steps may take some time to complete with little visible change in status. For example, the status *Processing database tables* may be shown for a significant amount of time, especially when upgrading from LANSA Composer version 4.0. Please allow the *LANSA Composer Server Setup* wizard adequate time to complete normally.

You may notice some other command windows briefly open and close during the installation. In addition the *LANSA Composer Version 5.0 Read Me* will open in your web browser. This is normal and you should simply wait for the *LANSA Composer Server Setup* wizard to finally report that the installation is complete. When it does so, continue with:

Completed the LANSA Composer Server Setup Wizard

# **Completed the LANSA Composer Server Setup Wizard**

The *LANSA Composer Server Setup* wizard reports that the installation is complete.



The LANSA Composer Server software is successfully installed.

The *LANSA Composer Server Setup* is in two parts. This completes the first part.

Next, the *Completing LANSA Composer Server Setup* wizard will start and guide you through the further steps necessary to load LANSA Composer files and data in order to prepare your LANSA Composer installation for first use. The following topics describe the steps and the responses you should provide at each step:

Completing LANSA Composer Server Setup LANSA Composer Files Location LANSA Composer Files Network Path Import Supplied Definitions Ready To Install Installing LANSA Composer Files and Data LANSA Composer Files and Data Have Been Installed Click *Finish* to close the wizard and then continue with the following: Completing LANSA Composer Server Setup

# **Completing LANSA Composer Server Setup**

When the *Completing LANSA Composer Server Setup* wizard starts you will see a screen like the one shown:



Click *Next* and continue with:

LANSA Composer Files Location

# **LANSA Composer Files Location**

The *Completing LANSA Composer Server Setup* wizard asks you to specify a location in the server file system for LANSA Composer files.

Completing LANSA	omposer Server Set	up		
ANSA Composer Fi Please specify a locatio	les Location	files.		
C:\LANSA Compo	ser\			Browse
This specifies a loc Composer clients ( example and tutori directories and file:	ation for files that are sl such as Transformation al files and a default loc	hared with the LA Map definitions) a ation for trading p	NSA as well as arther	
LANSA Composer exist.	Server Setup will create	e the folder if it do	es not	
	< <u>B</u> ack	<u>Next</u> >	Cancel	<u>H</u> elp

The LANSA Composer files location specifies the path of a directory that will be used by LANSA Composer for files that are shared with the LANSA Composer clients (such as Transformation Map definitions) and that will serve as a default location for trading partner directories and files.

The location you specify must be one that you can make available to LANSA Composer clients via a new or existing network share with appropriate read and write permissions. (The next step of the wizard will ask you to specify the network path to the same location.)

If you are upgrading from LANSA Composer version 4.0, use the value in item 12 in Complete your Upgrade Plan.

For a new installation, you may specify the name of a new or existing folder in your server's file system that you wish to use for this purpose.

The *Completing LANSA Composer Server Setup* wizard will (later) create the folder if it does not exist and will create a number of child folders and install shipped LANSA Composer files. The files that will be installed include:

- Files supporting LANSA Composer examples
- Master import files for supplied LANSA Composer definition
- Selected SEF files for use with EDI activities
- Files used in the LANSA Composer tutorials
- Master copies of files required for the Web Server support
- Master copies of files used with the LANSA Composer SOAP web service for running a Processing Sequence via the web service

When you have made your choice, click *Next* and continue with:

LANSA Composer Files Network Path

# **LANSA Composer Files Network Path**

The *Completing LANSA Composer Server Setup* wizard asks you to specify the network path that will refer to the same location that you specified in the previous step for LANSA Composer files.



The network path is important to allow the LANSA Composer client software (which is usually running on another computer) to access the location for LANSA Composer files. This must be correctly configured in order for certain functions of LANSA Composer (such as creating and modifying Transformation Maps) to operate correctly.

Depending on your choices, you may need to create a network share on your server in order to support this network access to the LANSA Composer files location. If necessary, consult your system administrator.

(The network share does not have to exist now in order to complete the LANSA Composer Server setup. But you must create it before you can begin to use the LANSA Composer client software – in particular, the Transformation Maps features. If it does not exist now, then you should make a note of the path used and ensure that you create it after the *Completing LANSA Composer Server Setup* wizard has finished.)

If you are upgrading from LANSA Composer version 4.0, use the value in item

13 in Complete your Upgrade Plan.

When you have made your choice, click *Next* and continue with: Import Supplied Definitions

# **Import Supplied Definitions**

The *Completing LANSA Composer Server Setup* wizard will import supplied definitions used with LANSA Composer to the LANSA Composer database.

Completing LANSA Composer Server Setup
Import Supplied Definitions         Supplied activity and other definitions will be imported.
Click Next to install LANSA Composer files and data and import definitions according to the choices you have made.
Import LANSA Composer examples
<ul> <li>System settings</li> <li>Code tables</li> <li>E vent handlers</li> <li>Configurations</li> <li>Document type and standard</li> <li>Transformations</li> <li>Processing sequences</li> <li>Trading partners</li> <li>Activities</li> </ul>
< <u>B</u> ack Next> Cancel <u>H</u> elp

The supplied definitions include system settings, code tables, configurations, activities and other items.

These items are necessary for a fully functional LANSA Composer installation and they will be imported, whether you are installing a new LANSA Composer system or upgrading a LANSA Composer version 4.0 system.

(Where you have made supported changes to these definitions in your existing installations, your changes will be preserved.)

Included amongst the supplied definitions are a set of examples, including example configurations of various sorts, example Transformation Map and Processing Sequences. You may choose whether to import the examples or not. To prevent import of the examples, uncheck the check box.

When you are ready, click *Next* and continue with:

Ready To Install

# **Ready To Install**

The *Completing LANSA Composer Server Setup* wizard is now ready to install the LANSA Composer files and definitions according to your choices in the previous steps.

eady To Install		
Ready to install LANSA Com	poser files and data	
LANSA Composer Serve	Setup will perform the following tas	:ks:
- Install LANSA Comp	oser files into C:\LANSA Composer\	4
- Set network path ("h	ome" path) to \\MYSERVER\LANS	SA Composer\
<ul> <li>Import LANSA Comp</li> </ul>	oser activity and other definitions	
- Import LANSA Comp	oser examples	
Click Next to proceed.		

If you wish to revise any of your choices, click the *Back* button to do so. When you are satisfied with your choices, click *Next* and continue with:

Installing LANSA Composer Files and Data

# **Installing LANSA Composer Files and Data**

The *Completing LANSA Composer Server Setup* wizard proceeds to install the LANSA Composer files and definitions as specified.

As it does so, you may see a number of status messages indicating the progress of the installation.

Completing LANS	A Composer Server Setup	X
Ready To Install Ready to install LAN	ISA Composer files and data	
LANSA Compose	er Server Setup will perform the following tasks:	
- Create folder	(s) for path C:\LC511B08\	
- Install LANS	A Composer files into C:\LC511B08\	
-Setne -Import -Import	Transform import in progress	
Click Next		
	< <u>B</u> ack <u>N</u> ext > Cancel	<u>H</u> elp

When the supplied definitions have been imported, you will see the *Import Log* window. It will indicate that the imports completed successfully.



There is no further action required here. Close the *Import Log* window to allow the installation to continue. When all installation steps have been completed, continue with:

LANSA Composer Files and Data Have Been Installed

# LANSA Composer Files and Data Have Been Installed

Both parts of the LANSA Composer Server setup are now complete.



Click Finish to close the Completing LANSA Composer Server Setup wizard.

Depending on your particular circumstances and choices there may remain some post-installations tasks that you should complete. For details, please continue with:

Complete the Installation
# **Repair an Existing LANSA Composer Installation**

Use this option to repair an existing installation of LANSA Composer version 5.0.

You might do this, particularly on instruction from LANSA support personnel, in the event that your installation of LANSA Composer Server has become damaged.

NOTE: As with any upgrade, you are strongly advised to make a backup of the LANSA Composer Windows server installation and the database that it uses before proceeding with the repair.



This option will re-install the LANSA Composer programs and other software. It will not change or remove your data describing your LANSA Composer solutions from the LANSA Composer database.

Follow the instructions shown by the *LANSA Composer Server Setup* wizard to proceed and complete the repair.

## **Complete the Installation**

If you were upgrading from LANSA Composer version 4.0, then you should first continue with the following:

Restore Configurations for LANSA for the Web applications Remove LANSA Composer Version 4.0

Remove LANSA Composer database tables from the earlier version

You may now need to complete some further steps to prepare your LANSA Composer server installation for use:

Apply Licenses

Configure Network Share for use by LANSA Composer Clients

Grant Database Permissions to the Web User

Configure IIS to Serve the LANSA Composer Web Components

Running the LANSA Composer Client on the Windows Server

Once the installation is complete, you will need to apply any server licenses that you have been issued for LANSA Composer. For more information refer to License the LANSA Composer Server.

## **Restore Configurations for LANSA for the Web applications**

This procedure applies if you are **upgrading** from LANSA Composer version 4.0. It does not apply if you are installing LANSA Composer Server.

You must complete these steps before you Remove LANSA Composer Version 4.0.

If you selected *Yes, serving other web applications too* for item 7 in the Complete your Upgrade Plan, then complete the following steps.

- 1. Stop Internet Information Services (IIS).
- 2. Locate the *l4w3serv.cfg* file in the LANSA Composer version 4.0 installation directory tree. For example, it might be found in *C:\Program Files\LANSA Composer Server\WebServer\IISPlugin.*
- 3. Copy the file to the equivalent location in the LANSA Composer version 5.0 installation directory tree, replacing the existing file.
- 4. Locate the *lroute.dat* file in the LANSA Composer version 4.0 installation directory tree. For example, it might be found in *C:\Program Files\LANSA Composer Server\Connect.*
- Copy the file to the equivalent location in the LANSA Composer version
   5.0 installation directory tree, replacing the existing file.
- 6. Start the *LANSA Communications Administrator* program using the shortcut provided in the LANSA Composer version 5.0 program group. Verify that the entries are correct for your other LANSA for the Web applications. Verify that the COMPOSER01 entry is present and identifies the listener port number for the LANSA Composer version 5.0 installation.
- 7. Start the *LANSA Web Administrator* program using the shortcut provided in the LANSA Composer version 5.0 program group. Open the active configuration and choose the *Maintain systems* command from the *Tools* menu. Then verify that the entries are correct both for the LANSA Composer version 5.0 installation and for the other LANSA for the Web applications.
- 8. Restart Internet Information Services (IIS).

When you have finished, continue with:

Remove LANSA Composer Version 4.0

## **Remove LANSA Composer Version 4.0**

This procedure applies if you are **upgrading** from LANSA Composer version 4.0. It does not apply if you are installing LANSA Composer Server.

The procedure to upgrade from version 4.0 treats Version 5.0 as a new installation, including specifying a new and different installation path for the LANSA Composer Version 5.0 software. This leaves the version 4.0 software on your system. Version 4.0 should now be removed.

There are two parts to removing the version 4.0 installation:

• **Delete the LANSA Composer version 4.0 shortcuts.** The program group and shortcuts for LANSA Composer version 4.0 are no longer required. You should delete the program group and any further shortcuts that you may have on the desktop or elsewhere.

#### • Delete the LANSA Composer version 4.0 installation directory. The LANSA Composer version 4.0 software remains on your system in its original installation directory – for example: *C:\Program Files\LANSA Composer Server*.

Refer to item 14 in Complete your Upgrade Plan. If you selected *No*, then you should:

• Delete (\*) the entire version 4.0 installation directory.

OR, if you selected *Yes*, then you should:

• Delete (\*) all the contents of the installation directory **except** the LANSA Composer Files directory (for example, *C:\Program Files\LANSA Composer Server\Data*)

(\*) When deleting the version 4.0 installation directory, do so by simply deleting the files and/or directories as instructed. Do NOT attempt to uninstall it using the LANSA (Selective removal) tool.

In both cases, be very careful that it is the version **4.0** shortcuts or installation directory that you are removing.

When you have finished, continue with:

Remove LANSA Composer database tables from the earlier version

# **Remove LANSA Composer database tables from the earlier version**

This procedure applies if you are **upgrading** from LANSA Composer version 4.0. It does not apply if you are installing LANSA Composer Server.

The database tables for LANSA Composer version 4.0 were contained in schema (or collection or owner) named F4XF3LIB.

The database tables for LANSA Composer version 5.0 are contained in schema (or collection or owner) named LC\_DTA.

The LANSA Composer Windows Server installer will have created the version 5.0 database tables and copied the version 4.0 data into them. However, the version 4.0 database tables contained in schema F4XF3LIB will remain in the database, even though they are no longer used.

You should use your database management tool (for example MS *SQL Server Management Studio*) to remove the schema F4XF3LIB and the database tables and views it contains.

Note: For future LANSA Composer versions, the schema name LC\_DTA will always be used and this procedure will no longer be necessary.

The following is a sample SQL script that you may be able to copy and adapt to your database management system to perform this task. (The specific syntax required for your DBMS may vary from the sample script shown below)

USE [database-name];

```
DROP VIEW [F4XF3LIB].[DXCVV02_V];
DROP VIEW [F4XF3LIB].[MWBHFL3_V];
DROP TABLE [F4XF3LIB].[DX02];
DROP TABLE [F4XF3LIB].[DX62];
DROP TABLE [F4XF3LIB].[DXAG];
DROP TABLE [F4XF3LIB].[DXAP];
DROP TABLE [F4XF3LIB].[DXAR];
DROP TABLE [F4XF3LIB].[DXAT];
```

DROP TABLE [F4XF3LIB].[DXCV];

DROP TABLE [F4XF3LIB].[DXDB]; DROP TABLE [F4XF3LIB].[DXDC]; DROP TABLE [F4XF3LIB].[DXDX]; DROP TABLE [F4XF3LIB].[DXEN]; DROP TABLE [F4XF3LIB].[DXEV];

DROP TABLE [F4XF3LIB].[DXFA1]; DROP TABLE [F4XF3LIB].[DXFA2]; DROP TABLE [F4XF3LIB].[DXFC]; DROP TABLE [F4XF3LIB].[DXHC]; DROP TABLE [F4XF3LIB].[DXJM];

DROP TABLE [F4XF3LIB].[DXL1]; DROP TABLE [F4XF3LIB].[DXL2]; DROP TABLE [F4XF3LIB].[DXL3]; DROP TABLE [F4XF3LIB].[DXL4]; DROP TABLE [F4XF3LIB].[DXL5]; DROP TABLE [F4XF3LIB].[DXL6];

DROP TABLE [F4XF3LIB].[DXLK]; DROP TABLE [F4XF3LIB].[DXM2]; DROP TABLE [F4XF3LIB].[DXM9]; DROP TABLE [F4XF3LIB].[DXMAPWORK];

DROP TABLE [F4XF3LIB].[DXMFCAT0]; DROP TABLE [F4XF3LIB].[DXMFCAT1]; DROP TABLE [F4XF3LIB].[DXMFCAT2]; DROP TABLE [F4XF3LIB].[DXMFCAT3]; DROP TABLE [F4XF3LIB].[DXMFCAT4]; DROP TABLE [F4XF3LIB].[DXMFCAT5]; DROP TABLE [F4XF3LIB].[DXMFCAT6];

DROP TABLE [F4XF3LIB].[DXMP]; DROP TABLE [F4XF3LIB].[DXMS]; DROP TABLE [F4XF3LIB].[DXNN]; DROP TABLE [F4XF3LIB].[DXNT];

DROP TABLE [F4XF3LIB].[DXP1]; DROP TABLE [F4XF3LIB].[DXP2]; DROP TABLE [F4XF3LIB].[DXP3]; DROP TABLE [F4XF3LIB].[DXP4]; DROP TABLE [F4XF3LIB].[DXP5]; DROP TABLE [F4XF3LIB].[DXP6]; DROP TABLE [F4XF3LIB].[DXP7]; DROP TABLE [F4XF3LIB].[DXP8];

DROP TABLE [F4XF3LIB].[DXPC]; DROP TABLE [F4XF3LIB].[DXPRTCTL]; DROP TABLE [F4XF3LIB].[DXPV];

DROP TABLE [F4XF3LIB].[DXR1]; DROP TABLE [F4XF3LIB].[DXR2]; DROP TABLE [F4XF3LIB].[DXR3];

DROP TABLE [F4XF3LIB].[DXSD]; DROP TABLE [F4XF3LIB].[DXSD02]; DROP TABLE [F4XF3LIB].[DXSK]; DROP TABLE [F4XF3LIB].[DXSM]; DROP TABLE [F4XF3LIB].[DXSP]; DROP TABLE [F4XF3LIB].[DXSRC256]; DROP TABLE [F4XF3LIB].[DXST]; DROP TABLE [F4XF3LIB].[DXSV]; DROP TABLE [F4XF3LIB].[DXSY];

DROP TABLE [F4XF3LIB].[DXT2]; DROP TABLE [F4XF3LIB].[DXT3]; DROP TABLE [F4XF3LIB].[DXT4]; DROP TABLE [F4XF3LIB].[DXT5]; DROP TABLE [F4XF3LIB].[DXT9]; DROP TABLE [F4XF3LIB].[DXTG]; DROP TABLE [F4XF3LIB].[DXTP]; DROP TABLE [F4XF3LIB].[DXTXT1];

DROP TABLE [F4XF3LIB].[DXX1EN]; DROP TABLE [F4XF3LIB].[DXX2IN]; DROP TABLE [F4XF3LIB].[DXX3GP]; DROP TABLE [F4XF3LIB].[DXX4MS]; DROP TABLE [F4XF3LIB].[DXX5AK]; DROP TABLE [F4XF3LIB].[DXXDCT]; DROP TABLE [F4XF3LIB].[DXXDSHF1]; DROP TABLE [F4XF3LIB].[DXXDST]; DROP TABLE [F4XF3LIB].[DXXDTR]; DROP TABLE [F4XF3LIB].[DXXECN];

DROP TABLE [F4XF3LIB].[DXXH]; DROP TABLE [F4XF3LIB].[DXXI]; DROP TABLE [F4XF3LIB].[DXXJ]; DROP TABLE [F4XF3LIB].[DXXSQN];

DROP TABLE [F4XF3LIB].[DXXTNLOG]; DROP TABLE [F4XF3LIB].[DXXTNREG]; DROP TABLE [F4XF3LIB].[DXXTNVLR];

DROP TABLE [F4XF3LIB].[EDI810IT1]; DROP TABLE [F4XF3LIB].[EDI810IT1P]; DROP TABLE [F4XF3LIB].[EDI810IT1R]; DROP TABLE [F4XF3LIB].[EDI810IT1S]; DROP TABLE [F4XF3LIB].[EDI810IT1T]; DROP TABLE [F4XF3LIB].[EDI810ITD]; DROP TABLE [F4XF3LIB].[EDI810MSG]; DROP TABLE [F4XF3LIB].[EDI810N1]; DROP TABLE [F4XF3LIB].[EDI810REF]; DROP TABLE [F4XF3LIB].[EDI810SAC]; DROP TABLE [F4XF3LIB].[EDI810TXI];

DROP TABLE [F4XF3LIB].[EDI850FOB]; DROP TABLE [F4XF3LIB].[EDI850MSG]; DROP TABLE [F4XF3LIB].[EDI850N1]; DROP TABLE [F4XF3LIB].[EDI850N1RF]; DROP TABLE [F4XF3LIB].[EDI850PER]; DROP TABLE [F4XF3LIB].[EDI850PO1]; DROP TABLE [F4XF3LIB].[EDI850PO1M]; DROP TABLE [F4XF3LIB].[EDI850REF]; DROP TABLE [F4XF3LIB].[EDI850TD5];

DROP TABLE [F4XF3LIB].[EDI855MSG];

DROP TABLE [F4XF3LIB].[EDI855N1]; DROP TABLE [F4XF3LIB].[EDI855PO1]; DROP TABLE [F4XF3LIB].[EDI855REF];

DROP TABLE [F4XF3LIB].[EDI997AK2]; DROP TABLE [F4XF3LIB].[EDI997AK3]; DROP TABLE [F4XF3LIB].[EDI997AK4]; DROP TABLE [F4XF3LIB].[EDI997MSG];

DROP TABLE [F4XF3LIB].[MWBDF]; DROP TABLE [F4XF3LIB].[MWBHF];

DROP TABLE [F4XF3LIB].[TUTORDH]; DROP TABLE [F4XF3LIB].[TUTORDL];

DROP SCHEMA [F4XF3LIB]; When you have finished, continue with: Apply Licenses

## **Apply Licenses**

Once the installation is complete, you will need to apply any server licenses that you have been issued for LANSA Composer. For more information refer to License the LANSA Composer Server.

When you have finished, continue with:

Configure Network Share for use by LANSA Composer Clients

## **Configure Network Share for use by LANSA Composer Clients**

In order to permit network access by the LANSA Composer clients to transformation map definition files (\*.mfd), you may need to configure a network share to certain folders used by the LANSA Composer Windows server installation.

**Note:** when configuring a network share for this purpose, the LANSA Composer clients will need both read and write access to the shared location. Look for *Advanced Sharing* and/or *Security* options in Windows in order to configure the security applying to the network share. If this is not configured properly, LANSA Composer clients may fail to save Transformation Maps correctly.

For more information refer to:

File locations relative to client

When you have finished, continue with:

Grant Database Permissions to the Web User

#### Grant Database Permissions to the Web User

To permit the LANSA Composer Web components (for example, the *Operations Console*) to operate successfully, you need to grant database permissions to the user id (for example, PCXUSER) specified in the User Id step of the *LANSA Composer Server Setup* wizard.

For a Microsoft SQL Server database, the *LANSA Composer Server Setup* wizard will have granted the permissions automatically.

Use your database management system's (DBMS) provided software to grant the permissions. You must grant the permissions in the database used for LANSA Composer – that is the database specified in the Setup the Local Database step of the *LANSA Composer Server Setup* wizard.

When you have finished, continue with:

Configure IIS to Serve the LANSA Composer Web Components

## **Configure IIS to Serve the LANSA Composer Web Components**

The LANSA Composer Windows server installation does not re-configure the LANSA for the Web IIS plug-in if it is already installed. This is deliberate, to avoid interfering with existing working LANSA for the Web applications, but it also means that you cannot immediately execute the LANSA Composer Web components (for example, the *Operations Console*) until you manually alter the IIS plug-in configuration.

There are several ways to do this. The following describes an *example* procedure to accomplish this.

You should refer to the applicable LANSA and IIS documentation for full details. The *example* procedure shows *one* way to configure the IIS plug-in for this purpose. The specific screens and the most appropriate configuration steps and values may differ on your system according to your circumstances and requirements.

The major steps in the *example* procedure are:

1. In IIS, add a new site binding for LANSA Composer

2. In the LANSA Web Administrator, add a data/application server for LANSA Composer

3. Copy LANSA Composer's web images, styles and script files

4. In LANSA Composer, alter System Settings to use the new port number

When you have finished, continue with:

Running the LANSA Composer Client on the Windows Server

#### 1. In IIS, add a new site binding for LANSA Composer

By default, the existing LANSA for the Web application is being served using the default port 80.

You can choose a new port number on which to serve the LANSA Composer Web components. Usually you should use a port number greater than 1000 to avoid conflict with "well-known" port assignments used for other protocols such as FTP.

Once you have chosen the port number (for example, 8080), perform the following steps to configure the web site to serve that port:

a. Open the Microsoft *Internet Information Services (IIS) Manager* and select the web site in which the LANSA for the Web IIS plug-in

(lansaweb) is installed.

- b. Open the *Site Bindings* window for the selected web site and click *Add* to add a new binding.
- c. In the *Add Site Binding* window, type the new port number.
- d. Make sure you have saved all your changes. There is no need to restart IIS services yet.



# 2. In the LANSA Web Administrator, add a data/application server for LANSA Composer

Perform the following steps to configure LANSA for the Web to invoke LANSA Composer to serve web requests made using the new port number:

- a. Open the *LANSA Web Administrator* and open the active LANSA web configuration (for example, *l4w3serv.cfg* in the *WebServer*\*IISPlugIn* folder of the LANSA system directory in which the IIS plug-in is installed)
- b. Select *Maintain Systems* from the *Tools* menu and click *Add*.

c. On the *Web Server* tab of the *Add LANSA System* window, type the *specific* port number determined above.

Advanced	H   H	old System	Reusable Sessions
Web	Server	Da	ta/Application Server
Web Serve	r		
Name:	<ul> <li>Any</li> <li>Specific</li> </ul>		
Port:	<ul> <li>C Any</li> <li>● Specific</li> </ul>	8080	

d. On the *Data/Application Server* tab of the *Add LANSA System* window, type the name of the *LANSA Communications Administrator* entry for LANSA Composer. Click the ellipsis (...) button to open the *LANSA Communications Administrator* if necessary. Type the user and password details necessary for LANSA for the Web to connect to the LANSA Composer application server.

	Web Serve	r Name	Port L	ANSA Syst	em LAN	NSA Library	Partition	Remote	
	<any></any>		80 ×	LOCAL				R	
	Ad	ld LANSA	System					×	
		Adva LANSA User:	Neb Server		System	Data/Applica	usable Session: tion Server	S	
_		Passw Confirm	ord: n Password:	******		Test	Connection		

e. Test the connection, save your changes and close the *LANSA Web Administrator*.

N S

f. Finally you should restart IIS now to cause the new configuration to take

effect.

# 3. Copy LANSA Composer's web images, styles and script files

The LANSA Composer Windows server installation program installed the web images, styles and script files for LANSA Composer in the *WebServer*\*Images* directories of the LANSA Composer installation location.

You will need to copy these files to the equivalent *WebServer\Images* directories of the LANSA system directory in which the IIS plug-in is installed.

The following files need to be copied:

- Image files named *xfd*\*.\* in *WebServer*\*Image* s
- Cascading style sheet files (.css) named *xfd\*.css* in *WebServer\Images\style*
- JavaScript files (.js) named *xfd\*.js* in *WebServer*\*Images*\*script*

# 4. In LANSA Composer, alter System Settings to use the new port number

Once IIS and LANSA for the Web are configured, you need to change the base URL used by LANSA Composer for the *Operations Console*.

To do this, start the LANSA Composer client and select *System Settings* from the navigator on the left. In the list of system settings, select *Base URL for browser interface*. In the *Value* edit box on the right, change the URL to use the new port number that has been bound to the LANSA for the Web IIS plug-in.



## **Running the LANSA Composer Client on the Windows Server**

When you install LANSA Composer Server on Windows, the LANSA Composer client software is also installed on the Windows server computer.

If you wish, and if your operating procedures permit it, you can logon to the Windows server computer and run the client software in offline mode (that is, accessing the LANSA Composer Windows server database directly).

This mode of running the LANSA Composer client is not recommended for normal use. Most usually, you will install the LANSA Composer client software on one or more client computers that will connect to this LANSA Composer server on Windows. To do this, follow the procedure described in LANSA Composer Client on Windows.

However, if you do choose to run the LANSA Composer client installed with the Windows server, there are some further steps that you must complete before you can do so.

• Install supporting software

Full use of the LANSA Composer client requires Optional Products that are not installed with the LANSA Composer server. You may need to manually install this software on the Windows server. If so, you can find the installation programs for Altova MapForce and the JDK in the \WINCLIENT\Installs directory of your LANSA Composer installation media.

# Services Used by the LANSA Composer Windows Server

The LANSA Composer Windows server is a complete application execution environment supporting client/server as well as web applications. As such, it has a number of services that must be active on the server to provide the necessary application support.

When the installation is completed into a new LANSA system, the necessary services are started for you and are usually configured to be automatically started whenever the server system is restarted.

In addition to the services described below, the web server instance used by LANSA Composer must be started in order to use web components such as the *Operations Console*.

#### The listener service

The LANSA listener provides communications services necessary to support both the client/server and web parts of LANSA Composer. The LANSA listener service must be active in order to start the LANSA Composer windows client or to open the LANSA Composer operations console in a web browser.

The LANSA listener service is usually identified with a service name beginning *LConnect Services*. If you have more than one LANSA product installed on your server, there may be more than one such service. In this case you can refer to the path specified in the service description to identify the listener service that is used by your LANSA Composer Windows server.

#### The Java Service Manager (JSM) service

The Java Service Manager service associated with the LANSA Composer installation must be started in order to use LANSA Integrator services with LANSA Composer. This is necessary for many of the activities shipped with LANSA Composer.

The Java Service Manager service is usually identified with a service name beginning *LANSA Integrator JSM Administrator Service*. If you have more than one LANSA product installed on your server, there may be more than one such service. In this case you can refer to the path specified in the service description to identify the listener service that is used by your LANSA Composer Windows server.

#### LANSA Composer Client on Windows

The latest information concerning this version of LANSA Composer is available on the LANSA web site. We recommend that you refer to this before proceeding with the installation or upgrade.

You will install the client on Windows in these steps:

Install LANSA Composer Client

Initialize Data and Settings

**Note:** If you are upgrading or replacing an existing installation of the LANSA Composer client software, it is strongly recommended that you uninstall the existing client software before beginning the installation of the new LANSA Composer client software. Refer to Uninstall LANSA Composer Client on Windows for instructions on how to do this.

## **Install LANSA Composer Client**

Log on to your Windows PC using a user that has administrative rights on the computer.

Insert the LANSA Composer installation DVD into the DVD drive on your Windows PC. After a short pause, the LANSA Composer installation program should load. (If the installer does not start automatically, use Windows Explorer to locate and open the *setup.exe* file in the root directory of the installation DVD.)



#### Start the setup program

Click the *Install LANSA Composer Client Software* option to start the LANSA Composer client setup program.

ANSA Composer Client Installati	on			
Composer	100			
Specify the directory into which LAN	SA Composer will be	installed		
Windows start menu folder				
LANSA Composer				
Create Deskton Iron				
LANSA Composer Server definition				
Communication settings				
Fully qualified name of server:	SERVER01		Port number:	4545
Optional products				
To enable full use of LANSA Compos	er features, this pro	gram may also ir	nstall the following so	ftware:
Altova MapForce 2014R2				
Altova MapForce is installed				
Java(TM) SE Development Kit 7				
Java Development Kit is installed				
LANSA Integrator Studio				
LANSA Web Administrator				
			_	
				OK Cancel

## **Complete your installation choices**

Complete the prompts in the *LANSA Composer Client Installation* window as follows:

#### **Installation directory**

Type the directory path into which LANSA Composer is to be installed. The setup program will create the folder if necessary.

The setup program will automatically generate and show the name for the start menu folder that will contain the program shortcuts based upon the installation directory path that you specify.

The setup program will create a set of icons in your Start menu for LANSA Composer. These icons will be contained in the start menu folder name shown.

It can also create icons on your desktop if you choose. Check the Create Desktop Icon box if you wish desktop icons to be created. If you choose this option, the setup program will create a single desktop icon (\*) that will always

be labelled LANSA Composer and that will start the LANSA Composer client software.

(\*) Note if you install LANSA Composer client software more than once, only one LANSA Composer desktop icon is created. If it already exists, it will be updated to refer to the last installed LANSA Composer client. If you wish to have more than one desktop icon for multiple LANSA Composer client installations, then you must manually copy and alter the desktop icons accordingly.

#### LANSA Composer Server Definition

Specify the DNS name or IP address of the LANSA Composer server to which you will connect and the communications port number used by the LANSA Listener for the specified LANSA Composer server.

(These details will be used to initialize the COMPOSER01 server entry in the LANSA Communications Administrator. The LANSA Composer client will default to using the COMPOSER01 entry for its server connection, although you can change this if necessary when you connect.)

#### **Optional Products**

Typical use of LANSA Composer client requires the installation of some other supporting software. The LANSA Composer client setup program can install this software for you:

- Altova MapForce is used to visually define transformation maps that transform data between formats such as database, XML, EDI and text. If you wish to define transformation maps then you must install Altova MapForce.
- A Java development kit at a supported level is required for compiling the Java code that is generated to implement your transformation maps
- The LANSA Integrator Studio may optionally be used for developing certain LANSA Integrator level solutions (such as web services solutions) that you might use with LANSA Composer. Usually, this type of activity will also require development activity using the LANSA development environment or other 3GL program languages.
- The LANSA Web Administrator is used to configure the LANSA for the Web support on your LANSA Composer server. You may need to use this program if you need or choose to change the default LANSA for the Web configuration used by LANSA Composer.

It is strongly recommended that you install this software if the setup program

suggests that you do so. If you do not, certain features of LANSA Composer (such as the transformation map features) will not be available to you unless you subsequently install them individually.

**Note 1:** If Altova MapForce is already installed at a version earlier than supplied with this version of LANSA Composer, you may wish to upgrade it to the supplied version. You can do so before or after installing LANSA Composer by directly running the appropriate installation executable file supplied on the LANSA Composer installation media. Refer to Mapping Tool Requirements for specific considerations that apply to upgrading from an earlier version of Altova MapForce.

**Note 2:** If you intend to use Altova MapForce to create mappings involving EDI X12, EDIFACT, HL7 or certain other specialized transaction standards, you may need to install additional configuration files for that transaction standard. These are not installed by the LANSA Composer client installation. The additional installers for EDI X12 and EDIFACT for the shipped version of Altova MapForce are provided on the LANSA Composer installation media. Configuration files for other versions or standards may be downloaded from the Altova web-site at http://www.altova.com/components\_mapforce.html.

## Proceed with the installation

When you have completed your installation choices, click *OK* to proceed with the installation and then click *Yes* when asked to confirm that the installation should proceed.



The setup program will proceed through a number of steps (according to your

choices) to install the supporting software and the LANSA Composer client. When complete, the *LANSA Composer Client Installation* window will close and the Read Me will be opened. You can now start the LANSA Composer client application using the shortcuts provided on your start menu. You should continue with the following steps:

• Initialize Data and Settings

## **Initialize Data and Settings**

The first LANSA Composer client to connect to the server should complete the initialization of the server system by importing supplied definitions and establishing system settings. To begin this task, start LANSA Composer using the shortcut provided and connect to your LANSA Composer server as described in Connect to LANSA Composer Server.

Now perform these two steps:

Step 1. LANSA Composer Server Initialization Step 2. Establish System Settings

#### Step 1. LANSA Composer Server Initialization

This step applies only when connecting to an IBM i server. If your LANSA Composer Server software is installed on a Windows software, you may disregard this section.

When the first LANSA Composer client starts and connects to the IBM i server, LANSA Composer will detect that the server system initialization has not been completed and prompt you to do so.

LANSA Composer Server Initialization	×
<ol><li>Supplied activity and other definitions must be imported.</li></ol>	
Click Finish to save your system settings and input the symplect definitions	
LANSA Composer Server Initialization	1
1. The following system settings must be correctly set.	
In order to successfully import supplied transformation maps, LANSA Composer needs to know the correct locations for transformation map definition files [".mid] and implementation files [".aii]	
LANSA Composer Server Initialization	
Composer	
Browse	
Welcome to LANCA Company	
weicome to LANSA composer.	
The LANSA Composer server database has not been initialized. Places councils the following states to council it is the council with the council with the council is the council of the following states are council in the council is t	
Prease complete the rollowing steps to complete the server initialisation. Lick Next to continue.	
h the application ettings, provides the	
	Help
el <u>H</u> elp	
	1
<u>Back</u> Next > Cancel <u>H</u> elp	

You should follow the prompts to complete the wizard and initialize the server system. In particular::

- When prompted to set selected system settings, you can refer to System settings for detailed information about the system settings whose values you are required to supply.
- When prompted to import supplied definitions, there is an option to import supplied examples. To prevent import of examples, uncheck the check box.

• Once done, click *Finish* to save your changes and proceed with the import.

## **Step 2. Establish System Settings**

During Step 1. LANSA Composer Server Initialization you may have specified certain system settings necessary for the basic initialization to complete. You should now review and revise as necessary all the remaining system settings for LANSA Composer. Certain features of LANSA Composer will not function correctly until you have specified appropriate settings for your installation.

To revise the system settings, expand *Administration and Housekeeping* and click *System Settings*. Refer to System settings for more information.

In particular, you must provide appropriate values for the following settings:

- Home path described in File locations
- Base URL for browser interface described in Browser
- URL for LANSA Integrator (JSM) console described in Browser.

# **Uninstall LANSA Composer Client on Windows**

The LANSA Composer client consists of the LANSA Composer client software and supporting software including the Mapping Tool (Altova MapForce) and the Java Development Kit. You can uninstall each of these components independently. If LANSA Composer client installed all these components, then you must separately uninstall each of them in order to completely remove LANSA Composer client software from the PC.

When uninstalling you should logon as the same user who installed LANSA Composer. If that is not possible, you should use a user that has administrative privileges on the computer.

# **Uninstall the Mapping Tool**

You can uninstall the Mapping Tool (Altova MapForce), if required using the *Add or Remove Programs* applet in *Control Panel*.

## Uninstall the Java Development Kit

You can uninstall the Java Development Kit, if required using the *Add or Remove Programs* applet in *Control Panel*.

## Uninstall the LANSA Composer client software

Follow these steps to uninstall the LANSA Composer Windows client software

1. Using Windows Explorer, locate and delete the LANSA Composer program folder in the Program Files folder on your PC. For example:

C:\Program Files\LANSA Composer

- 2. Locate and delete the LANSA Composer program folder on your *Start* menu. To do this:
  - a. Click the *Start* button
  - b. Click Programs
  - c. Right-click the LANSA Composer folder name
  - d. Select *Delete* from the popup menu.
- 3. Locate and delete the shortcut to the LANSA Composer program folder on your desktop, if present.
- 4. Using Windows' *Registry Editor*, locate the *LANSA Composer* key in the *Software* section of *HKEY\_CURRENT\_USER*. If you have more than one copy of LANSA Composer client installed, locate and delete ONLY the sub-

key that corresponds to the installed path for the copy that you are deleting. Otherwise you can delete the entire *LANSA Composer* key.

This step is optional, but it is important to do this if you intend to re-install LANSA Composer on the same computer.

Incorrectly editing the Windows registry can render your computer inoperable. You should only perform this procedure if you are well-versed in the Windows' registry and in using the Registry Editor.

## **Appendix B. License LANSA Composer**

The LANSA Composer server needs to be licensed for your server computer in order to design or execute business integration solutions. For more information refer to License the LANSA Composer Server.

The LANSA Composer client software does not need to be further licensed with the exception that you need to obtain and apply a license code for the mapping tool if you wish to design Transformation Maps. For more information refer to License the LANSA Composer Client.

**IMPORTANT NOTE:** 

If you are upgrading from LANSA Composer version 4.0, you must first request new LANSA Composer server licenses. Refer to Request a Server License for LANSA Composer.

# License the LANSA Composer Server

The LANSA Composer server needs to be licensed for your server computer in order to permit:

- LANSA Composer clients to connect to it
- LANSA Composer Processing Sequences to be run
- LANSA Integrator services to be used by supplied activities
- The Operations Console of LANSA Composer to be accessed from a webbrowser.

If you wish to use the CALL\_FUNCTION or COMPOSER\_RUN activities in their "remote" mode to call LANSA functions or run LANSA Composer Processing Sequences on a different server system, then the remote system will also need a *LANSA Composer Remote Request Server* license.

In order to license your server for LANSA Composer, you will need to:

Request a Server License for LANSA Composer

Apply the Server License for LANSA Composer

You may also refer to general information about licensing LANSA products at this web page:

#### **Product Licensing**

For additional information, you can also refer to About LANSA Composer Server Licenses.

### **Request a Server License for LANSA Composer**

The procedure you need to follow to request a server license for LANSA Composer depends on the type of server on which LANSA Composer will be installed. Refer to the appropriate heading below:

IBM i Server or Windows Server

#### IBM i Server

When you request a LANSA Composer license for your IBM i server you will need to provide the processor serial number, model number and feature code that apply to the server to be licensed.

You can obtain this information using the **WRKSYSVAL** command from an OS/400 Command Entry. On the *Work System Values* screen, use option 5 to display the required values:

- System serial number (QSRLNBR)
- System model number (QMODEL)
- Processor Feature (QPRCFEAT).

To display a single system value, you may also use the **DSPSYSVAL** command. For example, **DSPSYSVAL QPRCFEAT** will display the Processor Feature code.

#### Windows Server

To obtain the LANSA license codes required for your LANSA Composer server on Windows, you must supply the CPU details of the Windows server on which LANSA Composer is installed. This is achieved using a shipped LANSA utility, called  $x_cpu$ . Refer to the following web page for information on using  $x_cpu$ :

Using X\_CPU.EXE for generating CPU details for a License request

When you have run x\_cpu.exe and have located your x\_cpu\_<My cpu>\_<My *model*>.*txt* file, you can contact your LANSA vendor to request the license, providing the file and specify that you require one or both of:

- LANSA Composer license for a Windows server
- LANSA Composer Remote Request Server license for a Windows server

Refer to About LANSA Composer Server Licenses for information on the individual license codes required to license a LANSA Composer Windows server.

## Apply the Server License for LANSA Composer

The procedure you need to follow to apply a server license for LANSA Composer depends on the type of server on which LANSA Composer is installed. Refer to the appropriate heading below:

IBM i Server *or* Windows Server

#### IBM i Server

You will usually receive your IBM i server license as a file attachment to an email message. To apply the license, you need to follow these steps:

1. Save the license file on the IFS (integrated file system) of your IBM i server.

You might save it to the folder that was created for LANSA Composer. For example you might save it as:

#### /LANSA\_Composer/license.txt

- 2. Signon to an IBM i workstation or emulation session as QSECOFR or equivalent.
- 3. Type the following command and press Enter:

#### <pgmlib>/COMPOSER LICENSE

where **<pgmlib>** is the name of the program library for the LANSA system in which LANSA Composer is installed. For example, if you installed LANSA Composer into the default location, you would type:

#### LICPGMLIB/COMPOSER LICENSE

4. All current licenses, if any, will be displayed, along with their status. To apply your new license, proceed by following the instructions at the following web page:

Server License Codes - IBM i

5. Start or restart the LANSA listener, the HTTP server and the Java Service Manager (JSM). Refer to Subsystems and Jobs for information on controlling these subsystems and jobs.

#### Windows Server

You will usually receive your Windows server license as a file attachment to an e-mail message. To apply the license, you should follow the instructions at the following web page:

Server License Codes - Windows

# **About LANSA Composer Server Licenses**

A complete LANSA Composer server license is made up of at least four individual license codes that license separate components of LANSA and LANSA Composer functionality. The necessary license codes are delivered and applied using a single file. The following table describes the individual license codes that are necessary for the LANSA Composer server.

TFD	LANSA Composer run-time	Permits LANSA Composer Processing Sequences to be run.
LXX or Xnn	Client/server support (SuperServer)	Permits LANSA Composer clients to connect to the LANSA Composer server.
PWB, WEB or Wnn	Web support	Permits the HTTP server support necessary for the Operations Console of LANSA Composer to be accessed from a web-browser.
JME	LANSA Integrator	Permits LANSA Integrator services to be used by supplied activities.
EDI	LANSA Integrator	Permits EDI-specific LANSA Integrator services to be used by supplied activities.

If necessary you can check the status of individual licenses.

#### IBM i Server

To check the status of individual licenses on your IBM i server, type the following command and press Enter:

#### <pgmlib>/LANSA LICENSE

where **<pgmlib>** is the name of the program library for the LANSA system in which LANSA Composer is installed. For example, if you installed LANSA Composer into the default location, you would type:

#### LICPGMLIB/LANSA LICENSE

From the resulting menu, choose option 1 and press Enter to display LANSA
license status for the LANSA system.

#### Windows Server

To check the status of individual licenses on your Windows server, select the *LANSA License Status* item from the *Start* menu or LANSA shortcuts folder for the Visual LANSA system in which LANSA Composer is installed.

## LANSA Composer Remote Request Server License

If you wish to use the CALL\_FUNCTION or COMPOSER\_RUN activities in their "remote" mode to call LANSA functions or run LANSA Composer Processing Sequences on a different server system, then the remote system will also need a LANSA Composer Remote Request Server license. The following table describes the individual license codes that usually comprise a LANSA Composer Remote Request Server license.

TFQ	LANSA Composer remote request server	Permits the LANSA Composer Request Server to be executed in remote mode on the remote system.
LXX	Client/server	Permits LANSA Composer running on a separate
or	support	server system to connect to the LANSA Composer
Xnn	(SuperServer)	Request Server on the remote system.

## License the LANSA Composer Client

You need to request and apply a license code for the mapping tool if you wish to design Transformation Maps with LANSA Composer. You request the license code from your LANSA Composer supplier.

You will receive the license by email as a file attachment. Follow the instructions supplied in the email to apply the license to the LANSA Composer client your computer(s) to which it applies.

## Appendix C. The Mapping Tool

The Altova MapForce mapping tool has been selected for use with LANSA Composer specifically because of its powerful visual mapping of data between XML, EDI, Excel (OOXML), text, web service functions and database formats.

However, MapForce is a general-purpose mapping tool from a third-party vendor (Altova) and not designed specifically or exclusively for use with LANSA Composer. As a consequence, there are certain features and functions of MapForce that do not apply and/or that LANSA do not support in the context of LANSA Composer. Refer to Supported Functionality of the Mapping Tool for further details.

#### Also see

Learn about the Mapping Tool Mapping Tool Requirements Connecting to databases in Transformation Maps Supported Functionality of the Mapping Tool Supported Versions of the Mapping Tool

## Learn about the Mapping Tool

MapForce is a powerful visual mapping tool with a range of features and functionality that is beyond the scope of this guide to describe. You are strongly urged to complete the relevant tutorials and refer to the supplied MapForce documentation to acquaint yourself fully with this tool.

In particular, you should:

- Complete the LANSA Composer Tutorials.
- Complete applicable on-line training modules offered by Altova.
- Complete the MapForce tutorial which can be found in the Altova MapForce User and Reference Manual.
- Read the Altova MapForce User and Reference Manual.

A brief outline, Overview of the Mapping Tool, has been provided to give you a quick start with this tool.

**Note:** When completing the Altova on-line training modules, the MapForce tutorial or using other MapForce reference material, bear in mind the context and limitations for use of MapForce within LANSA Composer. For example, you may wish to disregard references to generating XSLT (or other languages) code since LANSA Composer uses MapForce to generate Java code for your Transformation Maps.

## **Overview of the Mapping Tool**

When you first edit a new Transformation Map, the Altova MapForce window is shown and the map workspace is empty.



Note that LANSA Composer has already assigned a file name and path to the map. The file name is based on the identifier you assigned to the Transformation Map in *Create or Copy a Transformation Map* in Work with Transformation Maps. You can see the file name in the title bar – in this case DEM\_ORDER.mfd. When you save your map, do NOT change the name or path. If you do so, LANSA Composer will not find or use your saved map definition.

A typical Transformation Map transforms data from one format to another – for example, from an in-coming order in xml format to one or more database tables. The steps to create this type of map will typically include:

#### 1. Insert the source component.

To insert an XML, EDI, text or database component that is to be the source component for the mapping, select the appropriate option from the *Insert* menu (or use the corresponding toolbar button) and complete the resulting dialogs to make your selection.



#### 2. Insert the target component.

Insert the XML, EDI, text or database component that is to be the target component for your map. To do this, you follow the same procedure as you did to insert the source component. When you have completed your selections, the components are added to the map workspace. You can rearrange the components in the workspace to make them easier for you to work with.



The source and target are shown in a tree view. The triangles in the edges of each item indicate the input and output connectors for use when mapping the individual elements, attributes, columns or fields.

For further information about inserting database components in your map, refer to Connecting to databases in Transformation Maps.

## 3. Connect elements, attributes, columns or fields of the source and target components.

Create connections between corresponding elements, attributes, columns or fields of the source and target components. To create the connections, you click and drag from the output connector (the small triangle) of a source data item to the input connector of the corresponding target data item. When you release the mouse over the input connector of the target data item, the mapping tool draws the connector between the two. You proceed in this way until all necessary connections are made.



#### 4. When complete, Save your map and close MapForce.

When you save your Transformation Map definition in MapForce, it is saved as a .mfd mapping file that contains the map definition. The .mfd file is not executable. In order to execute the map, you must prepare the map. Refer to Prepare Transformation Map for information.

**Note:** The mapping tool includes menu options and tool buttons for generating code for mapping in various languages. Do not use these options as LANSA Composer will generate the Java code for the map when you complete the *Prepare* step.

## **Mapping Tool Requirements**

To use the mapping tool and the Transformation Map features of LANSA Composer, you need the following on the workstation at which the Transformation Maps will be designed.

- LANSA Composer client software
- Altova MapForce installed, configured and licensed
- A supported Java Development Kit for compiling the Java code for the Transformation Map.

The Java development kit (JDK) must be at a level that is capable of compiling the Java code for Transformation Maps to the required Java version as required by the JVM on the LANSA Composer Server. In order to run Transformation Maps, the following minimum JVM level is required on the LANSA Composer server:

- For Transformation Maps created with Altova MapForce 2013 or later, your JVM must be at level 1.6 or higher.
- For Transformation Maps created with Altova MapForce 2009 or later, your JVM must be at level 1.5 or higher.
- For Transformation Maps created with Altova MapForce 2008 release 2 or earlier, your JVM must be at level 1.4 or higher. For Excel 2007 (OOXML) support, Java 1.5 or higher is necessary.

The LANSA Composer client setup will install these components if necessary.

#### Notes:

LANSA Composer will operate successfully with all recent versions of Altova MapForce. It is not mandatory to upgrade Altova MapForce to the version supplied with this version of LANSA Composer, although we strongly advise that you do so in order to take advantage of new functionality in Altova MapForce.

You should make sure that you use the same version of Altova MapForce on all LANSA Composer client installations.

If you intend to use Altova MapForce to create mappings involving EDI X12, EDIFACT, HL7 or certain other specialized transaction

standards, you may need to install additional configuration files for that transaction standard. These are not installed by the LANSA Composer client installation. The additional installers for EDI X12 and EDIFACT for the shipped version of Altova MapForce are provided on the LANSA Composer client media. Configuration files for other versions or standards may be downloaded from the Altova web-site at http://www.altova.com/components\_mapforce.html.

## **Connecting to databases in Transformation Maps**

The mapping tool in LANSA Composer supports mapping data between the various forms in which you exchange, transport, store and process it - for example, between XML and CSV.

Because most corporate data is stored in relational databases, a common application of Transformation maps is to map data between one of those formats and an application database. For example, in an application that permits trading partners to submit sales orders in an agreed XML format, your application might need to:

- transform incoming sales order XML data received from your trading partner into your received-orders database;
- transform order acknowledgement information from your database into an agreed XML form to send back to the trading partner.

The mapping tool supports the use of database components as either the source or target (or both) of the transformation for the databases it supports. These include:

- MS SQL Server
- Oracle
- IBM DB2
- MySQL

Other databases may be used if they provide suitable connectivity. You should refer to MapForce documentation and Altova for details of database support in MapForce.

## **Database Connectivity**

You must be able to establish two types of connectivity to a database for use with the mapping tool and the maps it generates:

• ODBC, ADO or JDBC connection from the LANSA Composer client computer

While defining your map you need to be able to establish an ODBC, ADO or JDBC connection (depending on the specific database type). This connection is initiated from the client computer running MapForce. Refer to:

Connecting to databases while defining a Transformation Map using ODBC or ADO

Connecting to databases while defining a Transformation Map using JDBC

• JDBC connection from the LANSA Composer server computer

When the Transformation map runs, it will use a JDBC connection to establish the connection to the database. This connection is initiated from the server computer on which the transformation map executes. Refer to:

Connecting to Databases while Executing a Transformation Map using JDBC

The following topic provides additional information about connecting to databases in Transformation Maps:

Database Connectivity Components and Drivers

The following topics provide examples, sample screens and guidance for connecting to selected specific database types:

Example Database Connections for IBM DB2 for i5/OS Example Database Connections for Microsoft Access Example Database Connections for Microsoft SQL Server Example Database Connections for Oracle

#### **Connecting to databases while defining a Transformation Map using ODBC or ADO**

You define your Transformation Map using the mapping tool. If you wish to use a database in your map, you do so by inserting a database component and completing the screens presented by the mapping tool that define the database connection. Because the mapping tool runs on your LANSA Composer client computer:

- The database connection is relative to (from) the LANSA Composer client computer
- Any required database connectivity components must be installed on the LANSA Composer client computer

Refer to the following topics for more information about inserting a database component using the mapping tool:

Insert a Database Component Select a Database Type Select Database Drivers Select or Create a DSN Select Database Schema and Tables

#### About ODBC

ODBC stands for open database connectivity. It is an API for using database management systems that is intended to be independent of programming languages, databases and operating systems. It provides methods for querying and updating data in a database.

#### **ODBC** Drivers

An ODBC driver is software that conforms to the ODBC standard and will provide the bridge between ODBC database access requests issued by the application (in this case the mapping tool, Altova MapForce) and a particular vendor's database system.

The driver accepts requests according to the ODBC standard API and processes them according to the particular database type or system for which it is designed.

Major database vendors provide approved ODBC drivers for their database products. Refer to Database Connectivity Components and Drivers for further information.

You can see the ODBC database drivers installed on your client computer by starting the *Data Sources (ODBC)* control panel applet and selecting the *Drivers* tab.

Name	Version	Company
Microsoft Access Driver (* mdb)	4 00 6304 00	Microsoft (
Microsoft Access-Treiber (* mdb)	4 00 6304 00	Microsoft (
Microsoft dBase Driver (* dbf)	4.00.6304.00	Microsoft (
Microsoft dBase VFP Driver (*.dbf)	6.00.8167.00	Microsoft (
Microsoft dBase-Treiber (*.dbf)	4.00.6304.00	Microsoft (
Microsoft Excel Driver (*xls) 4.00.6304.00 Microsoft		
Microsoft Excel-Treiber (* xls)	4.00.6304.00	Microsoft (
Microsoft FoxPro VFP Driver (*.dbf)	6.00.8167.00	Microsoft (
Microsoft ODBC for Oracle	2.575.1117.00	Microsoft (-
· · · · · · · · · · · · · · · · · · ·		È

#### **DSN (Database Source Name)**

While an ODBC driver knows how to query and update data in a particular type of database system, it usually requires additional information that refers it to a particular instance of that database system. Such information may include, for example:

- The name or address of the server hosting the database system
- Communications protocol information
- The database name or library name
- User authentication information

Such information is usually held in a DSN (database source name). You have to create the DSN's necessary to access your databases according to your own

database implementations. In Microsoft Windows, you can create three types of DSN:

- User DSN: these DSNs apply only to the computer on which they are created and are accessible only to the current user
- System DSN: a system DSN also applies only to the local computer but is visible to all users
- File DSN: a file DSN exists in a file in the file system and can be shared by users on multiple computers who have the same drivers installed.

You can see, change and create DSNs on your system by starting the *Data Sources (ODBC)* control panel applet and selecting the *User DSN, System DSN* or *File DSN* tabs. Most applications that support ODBC (including the mapping tool) will allow you to create or change a DSN from within the application in the course of establishing a database connection.

In either case, when you create or change a DSN, additional screens will be presented that will ask you for the information necessary to connect to a particular instance of the database. The screens are defined by the ODBC driver, depend on the requirements of the target database and will vary from one database driver to another.

#### **Insert a Database Component**

To insert a database component in the mapping tool, select the *Database* option from the *Insert* menu (or use the corresponding toolbar button) and complete the resulting dialogs to make your selection.



The dialogs you have to complete depend on the type of database connection you insert and on choices you have made in previous sessions in the mapping tool. The following topics describe some of the most common and important dialogs and choices:

Select a Database Type Select Database Drivers Select or Create a DSN Select Database Schema and Tables

## Select a Database Type

If you use the connection wizard when you insert a database component in the mapping tool, the wizard will ask that you select the type of database with which you wish to work.

Select a Database	
Connection Wizard	Connection Wizard
Existing Connections	Please select a source database and click on next. Alternatively you can select a source database and configure the wizard behavior. The wizard will have to be configured initially for each database.
<b>ADD</b>	C Microsoft Access ( AD0 )
ADO Connections	C Microsoft <u>S</u> QL Server (ADD )
sid	C <u>O</u> racle ( ODBC )
ODBC	C MySQL (ODBC)
UDBL Connections	● IBM <u>D</u> B2(ODBC)
	C Sybase ( ODBC )
	< <u>Back</u> <u>Next&gt;</u> <u>Close</u>

If you wish to connect to a IBM i database, choose *IBM DB2* and click *Next*.

## **Select Database Drivers**

When you insert a database component in the mapping tool, it may ask you to select the ODBC database driver to be used to establish the connection from the client computer:

tion Wizard	Edit the list of known drivers for the database.	
		Back to Configuration Page
ns	X Urivers	
	Microsoft ODBC for Oracle	
	Adaptive Server Anywhere 9.0	
	Client Access ODBC Driver (32-bit)	
l	iSeries Access ODBC Driver	
l	Adaptive Server Anywhere 8.0	
ections	Selected Drivers for the database:	
	iSeries Access ODBC Driver	

If you wish to connect to a IBM i database, choose the *IBM i Access ODBC Driver* or an equivalent driver that provides access to IBM IBM i databases. If you do not see this screen, and you need to select a different database driver, you can click the *Edit Drivers* button when you are asked to Select or Create a DSN.

## Select or Create a DSN

When you insert a database component, the mapping tool may ask you to select or create a DSN that is to be used to establish the connection from the client computer:

Connection Wizard     Select an option how you wish to connect to the database and click Next.     Existing   Connections   Image: Connections   CDBC Connections     Image: Connections <th></th> <th>Configure: IB</th> <th>M DB2</th> <th></th> <th></th>		Configure: IB	M DB2		
<ul> <li>Create a new Data Source Name (DSN) with the driver;</li> <li>Series Access ODBC Driver</li> <li>Use an existing Data Source Name:</li> <li>User DGN System DGN</li> <li>Edit Drivers</li> </ul>	Connection Wizard	Select an option how yo	ou wish to connect to the	database and click I	Vext.
Existing Connections   ADO Connections   DBC Connections     Skip the configuration step for wizard		Create a new Data S	Source Name (DSN) with	the driver:	
Connections   ADO Connections   DDBC Connections     Skip the configuration step for wizard	Existing	iSeries Access OD	BC Driver	•	
Image: Non-Sections       Edit Drivers         DDBC Connections       Edit Drivers         Image: Skip the configuration step for wizard       Edit Drivers	Connections	C Use an existing Data	a Source Name:		
ADO Connections DDBC Connections Skip the configuration step for wizard	ADO	O User DSN C	) System DSN		Edit Drivers
DBC Connections	ADO Connections				
IDBC Connections	-				
IDBC Connections	15				
Skip the configuration step for wizard	DBC Connections				
Skip the configuration step for wizard					
Skip the configuration step for wizard					
		Skip the configuratio	n step for wizard		
	l				-

If you have previously created a suitable DSN you can select it and use it again.

If no suitable DSN exists, you can create one. If you choose this option, further dialogs will be displayed that will ask for DSN details according to the selected database and driver. These will vary according to the type of database and driver.

For example, if you are connecting to IBM DB2 for i5/OS using the IBM i Access ODBC Driver, you will see a screen like this that prompts you for a name, description and other details for the new DSN:

iSeries Access for Windows ODBC Setup	? ×
General Server Packages Performance Language Catalog Translation Diagnostic	
Data source pame:	
Description:	
iSeries Access for Windows ODBC data source	
System:	
LANSA01 Connection Options	
OK Cancel <u>Apply</u> Help	,

Complete the details as required for your database and server. If in doubt, ask your system or database administrator for instructions.

If you are connecting to IBM DB2 for i5/OS using the IBM i Access ODBC Driver, in the simplest case you will need only to enter a data source name, description and select or specify the *System* to which you wish to connect. However, depending on your specific system and database configuration you may need to complete or change settings on other pages of the *IBM i Access for Windows ODBC Setup* screen.

Depending on the ODBC driver you chose and the configuration options specified in the DSN, you may be prompted for login information. If so, complete the login information to establish the connection:



## Choose to Connect Natively or via the ODBC API

The mapping tool may ask whether you wish to connect "natively" or "via the ODBC API".

How Do You Want the Datasource to Communicate with the Server?	×
<ul> <li>Natively</li> </ul>	
This is the recommended method for communicating with the server and for retrieving information. It generally provides far richer details about the schema of your database than can be obtained through ODBC alone, and provides additional functionality that ODBC cannot support (e.g. databases which have native XML schema support) but at the cost of being highly coupled to specific versions of the server.	
C Via the ODBC API	
If you experience problems retrieving information about your database(s) with the default method, or if for some other reason you would prefer to use the ODBC API strictly, choose this method. It may also provide a workaround to unexpected issues with the database (e.g. if the system tables have nonstandard names, or if you're using a newer version of a database which has not yet received official support).	
OK Cancel	

In most cases you should accept the default "native" selection.

You should only select "via the ODBC API" when instructed to by support or otherwise when experiencing difficulties using the native connection.

Note 1: "Natively" refers to Altova MapForce's native implementation in which it uses the ODBC API to directly interrogate system tables in supported target databases for information that is not normally accessible through the ODBC API alone. This implementation permits the mapping tool to implement advanced database functionality.

Note 2: Whichever choice you make here affects only the connection from the client while defining the map in the mapping tool. At runtime, the prepared map will always use JDBC connections using a LANSA Composer database configuration that you specify.

#### **Select Database Schema and Tables**

When you have selected the driver, DSN and provided connection information, the mapping tool establishes the connection and then allows you to choose the particular tables you wish to use in your transformation map:

Select Tables, Views to Insert		
Source		
♥ ♥ ♦		
E LANSAComposer1		
System Tables		
		8
Preview-		
Preview Table: TUTORDH		
	ΠΚ	Cancel

When you have selected the tables and clicked OK, the database component is inserted in the workspace for the map:



You can then proceed to insert further components and create connections between them. For further information, refer to Edit Transformation Map.

## **Connecting to databases while defining a Transformation Map using JDBC**

With MapForce 2012, you also have the alternative of creating connections via JDBC. It is important to note that this JDBC connection is only used inside MapForce and is not the same as the one that is used by LANSA Composer server at runtime.

To use JDBC, you will need the proper JDBC driver installed on your machine.

#### Finding the Right JDBC Driver

Depending on the database being used, you will need an appropriate JDBC driver. Please consult your system administrator to determine the appropriate driver to use. For example, to connect to IBM DB2 for i5/OS, you would usually use the JDBC driver provided by IBM in the file jt400.jar.

#### **Installing the JDBC Driver**

To be able to use a JDBC driver, it must first be added to the java CLASSPATH. To do this:

- 1. First put it somewhere on your system e.g. C:\Drivers.
- 2. Then go to Control Panel -> Systems -> Advanced.
- 3. Click on Environment Variables.

System Properties	<
System Restore Automatic Updates Remote	
General Computer Name Hardware Advanced	
You must be logged on as an Administrator to make most of these changes.	
Performance	
Visual effects, processor scheduling, memory usage, and virtual memory	
<u>S</u> ettings	
User Profiles	
Desktop settings related to your logon	
<u>Settings</u>	
C Startup and Recovery	
System startup, system failure, and debugging information	
Settings	
Environment Variables Error Reporting	
OK Cancel Apply	

4. Select the CLASSPATH variable and click edit.

nvironment Variabl	es	? ×
- User veriables for E	vieb	
Variable TEMP TMP	Value C:\Documents and Settings\Erick\Local C:\Documents and Settings\Erick\Local	
	<u>N</u> ew <u>E</u> dit <u>D</u> elete	
System variables	Value	
CLASSPATH	C:\drivers\db2icc-9.5.iar:C:\drivers\db	
ComSpec FP_NO_HOST_C include JAVA_HOME	C:\WINDOWS\system32\cmd.exe NO C:\Documents and Settings\Erick\My Do C:\Program Files\Java\jdk1.6.0_11	- -
	New Edit Delete	
	OK Can	cel

- 5. Add the location of the JDBC driver to the path.
- 6. You may need to restart your PC for the setting to take effect.

#### Using JDBC in MapForce

To opt to use JDBC, first you need to click JDBC Connections on the 'Select a Database' dialogue. You will then be presented with a screen similar to the following. Enter the appropriate details and you should be able to connect to the database.

For example, if you are connecting to an IBM DB2 for i5/OS database and you are using the JDBC driver jt400.jar, you would be selecting the AS400JDBCDriver as below and you would probably use a connection string similar to: 'jdbc:as400://{server-name}'.

😢 Select a Database	2	×
Connection Winnerd	JDBC Con	nections
Connection wizard	Enter a connection :	string and select a valid JDBC driver. Click on 'Connect' to proceed
6	Driver:	com.ibm.as400.access.AS400JDBCDriver
Existing Connections	Username:	
ADO Connections	Password:	
	Database URL:	
ODBC Connections		
JDBC Connections		
Global Resources		<u>C</u> onnect <u>Close</u>

#### **Connecting to Databases while Executing a Transformation Map using JDBC**

After you have defined and prepared your Transformation Map, you will execute it by including it in a Processing Sequence, probably along with other Activities, and then running the Processing Sequence. The Processing Sequence runs on your LANSA Composer server, and so:

- The database connection will be relative to (from) the LANSA Composer server computer
- Any required database connectivity components must be installed on the LANSA Composer server computer

When the Processing Sequence runs the Transformation Map, it does so by executing the Java code that was generated for the Transformation Map. If the Transformation Map includes any database components, the processing sequence controller prepares and provides a JDBC connection to the database(s).

The JDBC connection is prepared according to the Database Configuration that you provide as a parameter to the Transformation Map. You must define the Database Configuration with the correct details necessary to establish the particular database connection to be used.

Several example database configurations are supplied with LANSA Composer that illustrate how to connect to databases commonly used with LANSA Composer. You can copy and modify these example configurations or create your own from scratch.

Refer to the following topics for further information:

**Database Configuration** 

Transformation Maps

Database Connectivity Components and Drivers

**Note:** When your transformation map uses database components that connect to IBM DB2 for i5/OS database tables, LANSA Composer will remove the library qualifier from the generated Java code when you prepare the transformation map. You must setup your database configuration so that the required files can be found at run-time. Usually this is done by specifying the library name as part of the URL in the JDBC connection string. For more information, refer to

Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS.

#### About JDBC

JDBC stands for Java database connectivity. It is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

Major database vendors provide approved JDBC drivers for their database products. Refer to Database Connectivity Components and Drivers for further information.

## **Database Connectivity Components and Drivers**

When you insert a database component into a Transformation Map you must have the necessary ADO or ODBC database connectivity components installed on your client workstation. You define the map through the mapping tool and if you do a trial execute of the map in the mapping tool, you will use the selected ADO or ODBC database connector for the database connectivity.

When you execute the map in a LANSA Composer Processing Sequence, the map will use JDBC database connectivity components installed on your LANSA Composer server computer. The details for the JDBC database connection are provided by means of a Database Configuration.

LANSA Composer supplies neither the ADO or ODBC (client-side) nor the JDBC (server-side) database connectivity components. You must obtain and install these components yourself for use with LANSA Composer – usually these components are supplied by your database software vendor.

Most vendors will provide the requisite drivers on their installation CDs, as well as allow you to download them from their web site. In many case these drivers will be available at no charge. Most databases will need you to use a driver that is of exactly the same version as the database. Therefore, ensure that you have access to the correct driver for the version and type of database that you wish to connect to. Vendor web sites are the best source for this information.

Some useful sites are as follows for:

#### Oracle JDBC Drivers

#### Microsoft SQL Server Drivers

**Note:** For many databases, drivers can also be obtained from third party sources.

The IBM Toolbox for Java comes with two JDBC drivers for the IBM i , which can be used for accessing DB2 data:

- IBM Toolbox for Java JDBC Driver this is a Type 4 driver, that makes direct socket connect to the database host server
- IBM Developer Kit for Java JDBC Driver this is a Type 2 driver, and makes native method calls to the SQL CLI (Client Level Interface).

JDBC driver components need to be installed either into the jar directory of the LANSA Integrator instance or into a location that is accessible through the classpath.

## **Example Database Connections for IBM DB2 for i5/OS**

**Note:** example values and screens given in this section are based on particular database and driver software versions and configurations. They are provided for illustration and guidance. Your own environment may require different values and procedures.

You must be able to establish two types of connectivity to your IBM DB2 for i5/OS database for use with the mapping tool and the maps it generates:

• ODBC connection from the LANSA Composer client computer While defining your map you need to be able to establish an ODBC connection to the IBM DB2 for i5/OS database on your IBM i server. This connection is initiated from the client computer running MapForce. Refer to:

Connecting to IBM DB2 for i5/OS to define a Transformation Map

• JDBC connection from the LANSA Composer server computer

When the Transformation map runs, it will use a JDBC connection to establish the connection to the IBM DB2 for i5/OS database on your IBM i server. This connection is initiated from the server computer on which the transformation map executes and is defined in LANSA Composer by means of a Database Configuration. Refer to:

Connecting to IBM DB2 for i5/OS to execute a Transformation Map

For further important information concerning the use of IBM DB2 for i5/OS databases in Transformation Maps, please refer to:

Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS

# Connecting to IBM DB2 for i5/OS to define a Transformation Map

While defining a Transformation Map using the mapping tool, you will select the *Database* option from the *Insert* menu (or use the corresponding toolbar button) to insert a database component.



The dialogs you have to complete depend on the type of database connection you insert and on choices you have made in previous sessions in the mapping tool. The following steps describe some of the most common and important dialogs and choices: The actual windows you will see and have to complete in order to establish your database connection will vary according to the type of database, driver, previous database selections and other factors. What follows is an example sequence of windows for establishing a new connection to IBM DB2 for i5/OS running on a IBM i server.

1. On the first panel of the *Connection Wizard*, choose IBM DB2 as the database type and click *Next*.

(If the *Connection Wizard* is not shown, click the *Connection Wizard* button to display it)

Select a Database	
Connection Wizard	Connection Wizard
Existing	Please select a source database and click on next. Alternatively you can select a source database and configure the wizard behavior. The wizard will have to be configured initially for each database.
ADD Connections	<ul> <li>Microsoft <u>A</u>ccess (ADD )</li> <li>Microsoft <u>SQL</u> Server (ADD )</li> </ul>
ODBC Connections	<ul> <li>Oracle (ODBC)</li> <li>MySQL (ODBC)</li> <li>IBM DB2 (ODBC)</li> </ul>
	C Sybase (ODBC)
	< <u>B</u> ack <u>Next</u> > <u>C</u> lose

2. When prompted to select database drivers, choose the *IBM i Access ODBC Driver* or an equivalent driver that provides access to your IBM i database.

**Note:** The IBM i Access ODBC Driver needs to be installed on the PC that runs the LANSA Composer client and Altova MapForce to enable access to IBM DB2 for i5/OS database tables. Typically this is installed as part of the IBM IBM i Access installation.

Edit the list of known drivers for the database.	
	Back to Configuration Page
( 1511/075	
Microsoft ODBC for Oracle	<b>^</b>
Adaptive Server Anywhere 9.0	
Client Access ODBC Driver (32-bit)	
iSeries Access ODBC Driver	
Adaptive Server Anywhere 8.0	<b>▼</b>
	<u>•</u>
Selected Drivers for the database:	
iSeries Access ODBC Driver	
	Adaptive Server Anywhere 9.0 Client Access ODBC Driver (32-bit) iSeries Access ODBC Driver Adaptive Server Anywhere 8.0 Selected Drivers for the database: iSeries Access ODBC Driver

Normally you see this screen only on the first occasion that you choose the IBM DB2 option on the previous wizard panel. It associates specific drivers on your computer with the IBM DB2 option in the mapping tool. If you need to add or change drivers later, you can click the *Edit Drivers* button at the *Configure: IBM DB2* panel shown in the next step.

3. When prompted to configure IBM DB2, if you have previously defined a data source name (DSN) for use with LANSA Composer, choose it and click *Next*. Otherwise, click *Create a new Data Source Name (DSN)* and click *Next*.

Select a Database		×	
Connection Witterd	Configure: IBM DB2		
Connection wizard	Select an option how you wish to connect to the database and click Next.		
	Create a new Data Source Name (DSN) with the driver.		
Existing	iSeries Access ODBC Driver		
Connections	🔿 Use an existing Data Source Name:		
ADO	O User DSN O System DSN Edit Drivers		
ADO Connections			
ODBC Connections			
ODDC COnnections			
	okip the configuration step for wizard		
	< <u>B</u> ack <u>N</u> ext > <u>C</u> lose		

4. If you chose the option to *Create a new Data Source Name (DSN)*, you will now be prompted for details of the new DSN. The specific panels shown will vary according to the ODBC driver you chose. If you are connecting to IBM DB2 for i5/OS using the IBM i Access ODBC Driver, you will see a screen similar to this that prompts you for a name, description and other details for the new DSN:
| ries Access for Windows ODBC Setup  | ?    |
|---|------|
| General Server Packages Performance Language Catalog Translation Diagnostic |      |
| Data source name:   |      |
|   |      |
| Description:  |      |
| iSeries Access for Windows ODBC data source                                 |      |
| System:   |      |
| LANSA01 Connection Options  |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
|   |      |
| OV Canada Analy   | Hele |

Complete the details as required for your database and server. If in doubt, ask your system or database administrator for instructions.

If you are connecting to IBM DB2 for i5/OS using the IBM i Access ODBC Driver, in the simplest case you will need only to enter a data source name, description and select or specify the *System* to which you wish to connect. For example:

Data Source Name	iiiSRV400 This could be any meaningful name. It is the DSN name for this ODBC connection.
Description	IBM i Access for Windows ODBC Source
System	Specify the IBM i server system name (for example, the system name EARTH is used in many of these examples))

However, for more IBM DB2 for i5/OS considerations regarding the DSN, refer to Library names in the DSN.

However, depending on your specific system and database configuration you may need to complete or change settings on other pages of the *IBM i Access for Windows ODBC Setup* screen.

5. Depending on the ODBC driver you chose and the configuration options specified in the DSN, you may be prompted for login information. If so, complete the login information to connect to the database and continue with designing your transformation map.

Signon to i	Series	?×
	Server:	LANSA01
	<u>U</u> ser ID:	QPGMR
	Password:	
		OK Cancel

**Note:** The mapping tool may attempt to connect more than once to the database. This may result in you being prompted repeatedly for login information. You can avoid the multiple logins by changing the connection properties for the IBM i server in *IBM i Access for Windows* such that IBM i Access will prompt you for a password only for the first connection.

7. When your ODBC connection has been successfully established, the mapping tool allows you to choose the particular tables you wish to use in your transformation map:

Select Tables,Views to Insert
Source
Preview

8. Select the arrow symbol next to #LIBRARY to display IBM i libraries in a dropdown

Select Tables,Views to Insert
Source
#LIBRARY   #RPGLIB   ARCHIVE   AULT2F2   AVHSDATA   BPCSV61   BTF   CHM   CHM32E   CHMCDIRECT
Preview

5. Scroll down to find the library containing the files you want to use in your transformation map. For example, the tutorial database tables can be found in the library for the LANSA partition in which LANSA Composer was installed - for example, LICLICLIB.

ource	
▼ ▼ ♦	
DI III PSLMST	^
🗖 🎹 PSLSKL	
D III PSLTIMES	
🖸 🔟 SKLTAB	
🗖 📶 XBIG810	
🔲 🎹 XN01	~

6. Select the database tables you wish to use and click OK. A database component representing the selected tables is inserted in the workspace for the map.

**Note:** When used to access IBM DB2 for i5/OS database tables, the library from which you select the files is used only for the purpose of *defining* the transformation map. LANSA Composer will remove the library qualifier from the generated Java code when you prepare the transformation map. You must setup your database configuration such that the required files can be found in the desired library at run-time - this is described in Connecting to IBM DB2 for i5/OS to execute a Transformation Map.

# Connecting to IBM DB2 for i5/OS to execute a Transformation Map

When the Transformation map runs, it will use JDBC to establish the connection to the IBM DB2 for i5/OS database on your IBM i server. This connection is initiated from the server computer on which the transformation map executes.

LANSA Composer prepares the JDBC connection according to the Database Configuration that you provide as a parameter to the Transformation Map. You must define the Database Configuration with the correct details necessary to establish the particular database connection to be used.

#### **Example Database Configuration**

The following sample values assume that you are using native IBM JDBC driver to connect to the IBM DB2 for i5/OS database on your IBM i server:

Database driver class	Following is the class name used with the native IBM JDBC driver com.ibm.as400.access.AS400JDBCDriver
Database connection string	Following is a sample JDBC connection string to connect to the IB for i5/OS database running on your IBM i server using the native I JDBC driver:
	jdbc:as400://SYSNAME/LIBNAME;naming=sql;errors=fı format=iso; translate binary=true
	The following alternate form permits database tables from more th library (or schema) to be used. The JDBC connection property naming=system must specified for this to be effective:
	jdbc:as400://SYSNAME;naming=system;libraries=LIB1, LIB2,*LIBL;errors=full;date format=iso; translate binary=

#### **Further considerations**

- Refer also to Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS.
- When your transformation map uses database components that connect to IBM DB2 for i5/OS database tables, LANSA Composer will remove the library qualifier when you prepare the transformation map. You must setup your database configuration so that the required files can be found at run-

time. Usually this is done by specifying the library name as part of the URL in the JDBC connection string - in the first example shown, LIBNAME represents the library name.

- An alternate form of the connection string permits database tables from more than one library (or schema) to be used. For this form to be effective, the library name must be omitted from the URL portion of the connection string (so that the connection has no default schema) and naming=system must be specified. In this form, the libraries= property of the JDBC connection string specifies the libraries that are to be searched for the tables used in the Transformation Map. In the second example shown, LIB1, LIB2 and \*LIBL represent the library names to be used.
- The Database Configuration should specify the username and password of an i5/OS user authorized to perform the database transactions.
- Depending which driver version you are using, the IBM JDBC Driver is implemented in the file jt400.jar or jt400Native.jar. If it is not already present, you may need to copy the file to the /jar directory of your JSM instance and restart the Java Service Manager before using the driver with Transformation Maps in LANSA Composer. The jt400.jar is supplied with the IBM i operating system can usually and can usually be found in an IFS folder such as //QIBM/ProdData/OS400/jt400/lib
- Refer to relevant IBM reference materials for more information on the JDBC drivers available for IBM DB2 for i5/OS and the keywords and values that can be specified in the JDBC connection string for the native IBM JDBC driver. Following are some useful WEB references. Note that some of these references relate to specific versions of i5/OS and associated software. You should refer to the equivalent pages for the software versions you are using. IBM Toolbox for Java JDBC

IBM Toolbox for Java JDBC properties

IBM i - IBM Toolbox for Java: JDBC FAQ

ConnectionStrings.com - Forgot that connection string? Get it here!

## Additional Considerations for Transformation Maps Using IBM DB2 for i5/OS

The IBM i and its integrated environment including the IBM DB2 for i5/OS database offer some unique capabilities. Because the mapping tool is designed to support databases on a wide variety of platforms, some of the capabilities specific to the IBM i server present particular considerations for Transformation Maps that use IBM DB2 for i5/OS database components. The following topics describe some of these considerations:

Libraries and Library Lists

Using SQL SELECT Statements or SQL-WHERE Components in an IBM DB2 for i5/OS Database Components

Library names in the DSN

Transaction Control and IBM DB2 for i5/OS Database Journaling

**Note:** The considerations described are based on using Altova MapForce version 2010 release 3. Earlier (or later) versions of the mapping tool may be subject to different considerations or may differ with respect to the specific features mentioned.

#### **Libraries and Library Lists**

In a typical i5/OS environment, applications avoid hard-coded reference to specific libraries. This is a powerful feature that permits the same programs to be run using database tables and other objects in libraries that are determined by the run-time environment. LANSA Composer supports this feature in connection with Transformation Maps that use IBM DB2 for i5/OS database tables as follows:

1. Defining a Transformation Map that uses IBM DB2 for i5/OS database tables

When you define a Transformation Map that uses IBM DB2 for i5/OS database tables you select a specific library containing the files whose definitions you wish to use.

It is important to understand that the library from which you select the files is used only for the purpose of defining the transformation map. It does not mean that the Transformation Map will use the files in that library at execution time. 2. Preparing a Transformation Map that uses IBM DB2 for i5/OS database tables

When you prepare a Transformation Map that uses IBM DB2 for i5/OS database tables, LANSA Composer will remove the library qualifier from the generated Java code. This means that the Transformation Map can be executed using different instances of the database tables at run-time. This is the desired behavior in an IBM i server environment.

Note that LANSA Composer removes library qualifiers only for an IBM DB2 for i5/OS database component - not for any other type of database. It does so by using a temporary copy of the map definition (\*.mfd) file in which it has set MapForce's database component setting Strip schema names from table names (if not already set). The original map definition (\*.mfd) file is not changed in this process.

**Note:** LANSA Composer will not remove library qualifiers for certain types of map constructs. For more information, refer to Using SQL SELECT Statements or SQL-WHERE Components in an IBM DB2 for i5/OS Database Components.

3. Executing a Transformation Map that uses IBM DB2 for i5/OS database tables

When you execute a Transformation Map that uses IBM DB2 for i5/OS database tables, LANSA Composer prepares the JDBC connection according to the Database Configuration that you provide as a parameter to the Transformation Map. The Database Configuration specifies the library (or libraries) to use.

## Using SQL SELECT Statements or SQL-WHERE Components in an IBM DB2 for i5/OS Database Components

The mapping tool provides two features that allow you to enter SQL that will be executed by the Transformation Map:

- A database component may include one ore more SQL SELECT statements, the results of which may be used as "virtual" tables for mapping purposes.
- Another type of component, SQL-WHERE, allows you to filter database data conditionally using an SQL WHERE clause that you enter yourself.

When you use these features with an IBM DB2 for i5/OS database component, you must understand that the Strip schema names from table names feature does not strip schema (library) names from the user-defined SQL SELECT statement

or SQL-WHERE clause. This means that when you prepare the Transformation Map, any schema (library) qualifiers present will remain in the generated Java code.

In order to avoid creating Transformation Maps that access or update database tables in a library other than intended, you must avoid using schema (library) qualifiers in your SQL SELECT statement or SQL-WHERE clause when using these MapForce features.

There is a secondary consequence of this. If you omit or remove the library qualifiers from your SQL SELECT statement or SQL-WHERE clause, you may find that:

- You can no longer test execute the map in the mapping tool using the Output tab, and/or
- The Prepare step may fail while generating the Java code.

In both cases, this occurs because ODBC cannot find the files because the library or libraries containing the files are not specified in the DSN. If this occurs, it will be necessary to use an ODBC DSN that specifies an appropriate library list. Refer to Library names in the DSN for more information.

#### Library names in the DSN

When you define a Transformation Map that uses IBM DB2 for i5/OS database tables, MapForce prompts you to select a specific library containing the files whose definitions you wish to use.

As a result, it is frequently not necessary to specify the library or libraries you wish to use in the ODBC DSN you use for the connection. This can be convenient because it allows you to reference tables in multiple libraries using a single DSN.

However, when using certain MapForce features with such a DSN you may be affected by one or both of the following:

- You cannot successfully test execute the map in the mapping tool using the Output tab, and/or
- The Prepare step may fail while generating the Java code.

In particular this occurs:

- When you use MapForce SQL SELECT statements in which (as recommended) you have omitted the library qualifiers;
- When you use MapForce SQL-WHERE components in which (as

recommended) you have omitted the library qualifiers;

• When you explicitly select the Strip schema names from table names checkbox in the database component settings.

To avoid these issues, you are advised to use a DSN that specifies the library or libraries you wish to use. You can specify such a DSN for the IBM i Access ODBC Driver using the Server page of the *IBM i Access for Windows ODBC Setup* window in one of the following ways:

1. To specify a single library, type the library name in the SQL default library box as shown:

System i Access for Windows ODBC Setup	×
General Server Data Types Packages Performance Language Catalog Conversions Diagnostic	
Naming convention: SQL naming convention (*SQL) SQL default library: LICLICLIB	
Library list:	
Connection type:	
Use ODBC access mode     O Use ODBC access mode	
<ul> <li>Limit to Read/Call (Select and Call statements allowed)</li> <li>Limit to Read-Only (Select statements only)</li> </ul>	
Override default database with the following:	
Advanced	
OK Cancel <u>Apply</u> Help	

- 2. To specify a list of libraries:
  - a. Select System naming convention (\*SYS) in the Naming convention drop-down list (but see note below)
  - b. Specify the required libraries in the Library List box as shown:

**Note:** If you use the system naming convention, you will have to select the option to connect via the ODBC API (instead of Natively)

when prompted while adding the database component. Otherwise you will not be able to see or select your library name in the Add/Remove Tables window.

neral	Server	Data Types	Packages	Performance	Language	Catalog	Conversions	Diagnostic
Naming	conven	ition:		_				
System	n naming	convention (	*SYS)	<b>_</b>				
SQL de	fault libr	ary:						
Library i	ISC:							
APPDA	ATALIB I	LICLICLIB *US	RLIBL					
Connec	tion type	9.						
ΘU	se ODB	C access mod	le					
ΟU	se ODB	C access mod	le, Call allowe	ed				
O Li	imit to R	ead/Call (Sele	ct and Call st	atements allow	ed)			
O Li	imit to R	ead-Only (Sele	ect statement	s only)				
	erride del	fault database	with the follo	wina:				
								<u> </u>
				Advance	ed			

#### Transaction Control and IBM DB2 for i5/OS Database Journaling

You may use a database component, including an IBM DB2 for i5/OS database component, as the source or target of a mapping.

When used as the source of a mapping, you are using the database for read-only and transaction control considerations do not apply.

When used as the target of a mapping however, the Transformation Map may be performing insert, update and/or delete actions against the affected database tables, according to the Database Table Actions you specify in the mapping tool. In this case, you may wish to consider transaction control (commitment control) options that allow you to ensure that a transaction is either completed in its entirety or rolled back in the event of an error.

The mapping tool allows you to specify that you wish to use transactions at two levels:

- In the database *Component Settings* window, check the *Use Transactions* checkbox to indicate that transaction control is to be applied to database table actions for all tables used in the database component;
- In the *Database Table Actions* window, check the *Use Transactions* checkbox to indicate that transaction control is to be applied to database table actions for that table.

In most cases, specifying the option at the component level is advised. Unresolved issues have been observed when attempting to specify the option at the database table actions level (at the level of MapForce version 2012 release 2). Refer to the Altova MapForce documentation for more details concerning these options.

The remainder of this section is concerned with the particular considerations that apply when you use these options in connection with an IBM DB2 for i5/OS database component.

1. If you do not specify the *Use Transactions* options in your map, no special considerations apply. There is no need to have database journaling in effect for the affected IBM i database tables. (Note that this is a change from the advice that applied to earlier versions of LANSA Composer and the mapping tool).

However, if when you execute the map, you experience an error SQL7008 (<file> in library> not valid for operation) with reason code 3, it may be because your Database Configuration is not configured appropriately for this case. In particular, your Database Configuration:

- Should specify transaction isolation: None
- Should not contain keywords in the JDBC connection string that will cause the database manager to attempt to use commitment control. For example, make sure that it does not contain the string 'true autocommit=true'.
- 2. If you do specify the *Use Transactions* options in your map, then you will need to:
- a. Ensure that the database itself is configured to support the desired transaction control. This usually means at least that the database tables must be journalled. Depending on how your database tables were created, you may have to use IBM i commands such as CRTJRNRCV (Create Journal Receiver), CRTJRN (Create Journal) and STRJRNPF (Start Journal Physical File). Refer to IBM i documentation for details.

b. Specify appropriate values in your Database Configuration to support the desired transaction control. In many cases, the following values will be appropriate:

Commit No automatically Transaction Read uncommitted isolation

(The values suggested above were successfully tested with IBM i OS 7.1. The specific values required on your system may vary according to your environment and system and database configuration. The values required may also vary according to the JDBC driver level and the operating system level.)

When using transaction control you are strongly advised to test that the map options you specified along with the database configuration you propose to use do work as expected, both for the case where the transaction completes normally and for the case where an error occurs requiring the transaction to be rolled back. A transformation map that uses transaction control on an i5/OS database may appear to function correctly in normal circumstances. It may not become apparent until the necessity arises to invoke the transaction rollback that the database setup (eg: lack of journaling) and/or the database configuration are not appropriate for supporting the desired transaction control.

## **Example Database Connections for Microsoft Access**

**Note:** Example values and screens given in this section are based on particular database and driver software versions and configurations. They are provided for illustration and guidance. Your own environment may require different values and procedures.

You must be able to establish two types of connectivity to your Microsoft Access database for use with the mapping tool and the maps it generates:

• ODBC connection from the LANSA Composer client computer While defining your map you need to be able to establish an ODBC connection to the Microsoft Access database on your network. This connection is initiated from the client computer running MapForce. Refer to:

Connecting to Microsoft Access to Define a Transformation Map

• JDBC connection from the LANSA Composer server computer

When the Transformation map runs, it will use a JDBC connection to establish the connection to the Microsoft Access database on your network. This connection is initiated from the server computer on which the transformation map executes and is defined in LANSA Composer by means of a Database Configuration. Refer to:

Connecting to Microsoft Access to Execute a Transformation Map

### **Connecting to Microsoft Access to Define a Transformation Map**

While defining a Transformation Map using the mapping tool, you will select the *Database* option from the *Insert* menu (or use the corresponding toolbar button) to insert a database component.



The dialogs you have to complete depend on the type of database connection you insert and on choices you have made in previous sessions in the mapping tool. The following steps describe some of the most common and important dialogs and choices: The actual windows you will see and have to complete in order to establish your database connection will vary according to the type of database, driver, previous database selections and other factors. What follows is an example sequence of windows for establishing a new connection to a Microsoft Access database on your network.

1. On the first panel of the *Connection Wizard*, choose Microsoft Access as the database type and click *Next*.

(If the *Connection Wizard* is not shown, click the *Connection Wizard* button to display it)

Select a Database		×
Connection Wizard	Connection Wizard Please select a source database and click on next. Alternatively you can select a source database and configure the wizard behavior. The wizard will have to be configured initially for each database.	
ADO Connections	<ul> <li>Microsoft Access ( ADO )</li> <li>Microsoft SQL Server ( ADO )</li> <li>Oracle ( ODBC )</li> </ul>	
ODBC Connections	C MySQL (ODBC) C IBM <u>D</u> B2 (ODBC)	
Global Resources	< Back Next> Close	

**Note:** The Microsoft Access ODBC Driver need to be installed on the PC that runs Composer Client and Altova MapForce to enable access to Microsoft Access databases. Most standard Windows installations include this and other standard Microsoft ODBC drivers. If your PC does not have this driver, you may be able to obtain it by downloading and installing the appropriate Microsoft Data Access Components package.

2. When prompted to select the Microsoft Access database, type the network path to the database or use the *Browse* button to locate it. When complete, click *Next*.

Select a Database		×
Connection Wizard	Connect to MS Access	
Existing	Click on browse and select a Microsoft Access database. Then click on next to connect to the database.	
ADO Connections	\\LCWinSrv\Composer\Tutorial\TUTORIAL.mdb Browse	
CDBC Connections		
Giobal Resources		
L	< <u>B</u> ack <u>Next</u> Close	

3. When your ODBC connection has been successfully established, the mapping tool allows you to choose the particular tables you wish to use in your transformation map:

Insert Database Objects           Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.	×
Source	
▼ ▼ ◆ Add/Edit SELECT Statement Add/Edit Table Relations (0)	
Image: Tutorial ADO         Image: Tutorial         Image: Tuto	
Show Preview >> Insert Canc	el

4. When you have completed your selections, click *Insert*. A database

component representing the selected tables is inserted in the workspace for the map.

## **Connecting to Microsoft Access to Execute a Transformation Map**

When the Transformation map runs, it will use JDBC to establish the connection to the Microsoft Access database on your network. This connection is initiated from the server computer on which the transformation map executes.

LANSA Composer prepares the JDBC connection according to the Database Configuration that you provide as a parameter to the Transformation Map. You must define the Database Configuration with the correct details necessary to establish the particular database connection to be used.

#### **Example Database Configuration**

The following sample values assume that you will use the JDBC-ODBC bridge included with your JDK on the Windows server running LANSA Composer in order to connect to your Microsoft Access database. This approach requires that the machine also have the Microsoft Access ODBC driver installed. There are also other JDBC solutions available for Microsoft Access from third-party providers.

Database	Following is the class name used with the JDBC-ODBC bridge:
driver class	sun.jdbc.odbc.JdbcOdbcDriver
Database connection	Following is a sample JDBC connection string to connect to a Mic database using the JDBC-ODBC bridge:
string	jdbc:odbc:;DRIVER=Microsoft Access
	Driver (*.mdb);DBQ=C:\COMPOSER\TUTORIAL\TUTC

#### **Further considerations**

- The Database Configuration need not specify a username and password unless you have defined security in your Microsoft Access database.
- Following are some useful references related to the JDBC-ODBC bridge and Microsoft Access ODBC driver:

JDBC-ODBC Bridge Driver Type 1 Driver - JDBC-ODBC bridge Frequently Asked Questions about JDBC TechBookReport - JDBC-ODBC Bridge Tutorial ConnectionStrings.com - Forgot that connection string? Get it here!

## **Example Database Connections for Microsoft SQL Server**

**Note:** Example values and screens given in this section are based on particular database and driver software versions and configurations. They are provided for illustration and guidance. Your own environment may require different values and procedures.

You must be able to establish two types of connectivity to your Microsoft SQL Server database for use with the mapping tool and the maps it generates:

• ADO or ODBC connection from the LANSA Composer client computer While defining your map you need to be able to establish an ADO or ODBC connection to the Microsoft SQL Server database on your network. This connection is initiated from the client computer running MapForce. Refer to:

Connecting to Microsoft SQL Server to Define a Transformation Map

• JDBC connection from the LANSA Composer server computer

When the Transformation map runs, it will use a JDBC connection to establish the connection to the Microsoft SQL Server database on your network. This connection is initiated from the server computer on which the transformation map executes and is defined in LANSA Composer by means of a Database Configuration. Refer to:

Connecting to Microsoft SQL Server to Execute a Transformation Map

## **Connecting to Microsoft SQL Server to Define a Transformation Map**

While defining a Transformation Map using the mapping tool, you will select the *Database* option from the *Insert* menu (or use the corresponding toolbar button) to insert a database component.



The dialogs you have to complete depend on the type of database connection you insert and on choices you have made in previous sessions in the mapping tool. The following steps describe some of the most common and important dialogs and choices: The actual windows you will see and have to complete in order to establish your database connection will vary according to the type of database, driver, previous database selections and other factors. What follows is an example sequence of windows for establishing a new ADO connection to a Microsoft SQL Server database running on your network.

1. On the first panel of the *Connection Wizard*, choose Microsoft SQL Server as the database type and click *Next*.

(If the *Connection Wizard* is not shown, click the *Connection Wizard* button to display it)

Select a Database		×
Connection Wizard	Connection Wizard Please select a source database and click on next. Alternatively you can select a source database and configure the wizard behavior. The wizard will have to be configured initially for each database.	
ADO Connections	<ul> <li>Microsoft Access (ADO)</li> <li>Microsoft SQL Server (ADO)</li> <li>Oracle (ODBC)</li> <li>MySQL (ODBC)</li> <li>IBM DB2 (ODBC)</li> </ul>	
Global Resources	C Sybase ( ODBC ) < <u>Back</u> < <u>Dext&gt;</u> Close	

Note: These examples will use the Microsoft OLE DB Provider for SQL Server to establish an ADO connection to the MS SQL Server database. This is the preferred means of accessing MS SQL Server from the mapping tool. Refer to the MapForce documentation for more information.

It is also possible to establish an ODBC connection if you have an ODBC driver installed for MS SQL Server. To do so, click ODBC Connections and complete the steps for creating an ODBC connection. You may need to create a DSN for your server and database.

2. When prompted to select an ADO provider, choose the *Microsoft OLE DB Provider for SQL Server* and click *Next*.

Configure. MS	SQL Server		
Select a provider for the d	atabase and then click o	in next:	
Microsoft OLE DB Provid	er for SQL Server		
Skip the configuration st	ep for wizard		
	< <u>B</u> ack	<u>N</u> ext >	<u>C</u> lose
	Skip the configuration st	Microsoft OLE DB Provider for SQL Server	Microsoft OLE DB Provider for SQL Server         Skip the configuration step for wizard         < Back

installed, you can also select the *SQL Native Client* option from the drop-down list.

3. The Connection tab of the Data Link Properties window allows you to enter details of the database to which you wish to connect. Complete the details as described below:

🗟 Data Link Properties 🛛 🗙
Provider Connection Advanced All
Specify the following to connect to SQL Server data: 1. Select or enter a server name: TEST0101
<ul> <li>2. Enter information to log on to the server:</li> <li>C Use <u>Windows</u> NT Integrated security</li> <li>O Use a specific user name and password:</li> </ul>
User name: sa Password: ••••• Riank password  Allow saving password
3. Select the database on the server:
TUTORIAL     Attach a database file as a database name:
Using the filename:
Test Connection
OK Cancel Help

- Type the name of the server that hosts the MS SQL Server data or select it from the drop-down list. In the illustration, the server name is *TEST0101*.
- Complete the logon security credentials as required for your server and database. If you enter a required password, you must check the *Allow saving password* box.
- Select or type the name of the database on the server that you wish to use.

Important: If you are having connection problems at this point, then you need to check if the MS SQL Server is reachable (ping). If that is ok, check that the username / password is correct. If using "Windows NT Integrated Security", then you will need to make sure that the current Windows user of the client is registered as a valid user in the MS SQL Server. Refer to the security options on the Database for adding a new user on the MS SQL Server Guide. Additionally, for more information on the error, see the SQL Server logs in the SQL Server Management Studio.

You can click *Test Connection* to verify that the connection can be

established with the details you have entered. When complete, click *OK* to continue.

**Important:** If you enter a required password, you must check the *Allow saving password* box to save the password in the connection string. If a password is required, it must be saved in the connection string. Otherwise the connection will fail when MapForce attempts to insert the component, even if the *Test Connection* succeeds.

4. When your connection has been successfully established, the mapping tool allows you to choose the particular tables you wish to use in your transformation map:

🖙 Insert Database Objects		×
Please choose Tables, Views, or create your own SQL SELECT statements to insert into your mapping. SQL SELECT statements defined by the user are stored locally in the mfd file.		
Source		-
▼ ▼ ♦   Add/Edit SELECT Statement   Add/Edit Table Relations (0)		
	-	
	-	
Show Preview >> Insert Car	ncel	

Select the database tables you wish to use and click OK. A database component representing the selected tables is inserted in the workspace for the map.

## **Connecting to Microsoft SQL Server to Execute a Transformation Map**

When the Transformation map runs, it will use JDBC to establish the connection to the Microsoft SQL Server database on your network. This connection is initiated from the server computer on which the transformation map executes.

LANSA Composer prepares the JDBC connection according to the Database Configuration that you provide as a parameter to the Transformation Map. You must define the Database Configuration with the correct details necessary to establish the particular database connection to be used.

#### **Example Database Configuration**

The following sample values assume that you are using Microsoft's JDBC driver to connect to the Microsoft SQL Server database on your network:

Database driver	Following are class names commonly used with the Microsoft SQI Server JDBC drivers:
class	For Microsoft SQL Server 2000: com.microsoft.jdbc.sqlserver.SQLServerDriver
	For Microsoft SQL Server 2005: com.microsoft.sqlserver.jdbc.SQLServerDriver
Database connection string	Following is a sample JDBC connection string to connect to the Microsoft SQL Server database named TUTORIAL running on the server named TEST0101:
	jdbc:sqlserver://TEST0101:1433;databasename=TUTORI/

#### **Further considerations**

• The Database Configuration should specify the username and password of the Windows or SQL Server user authorized to perform the database transactions.

If necessary, you can download the Microsoft SQL Server JDBC Driver 3.0 from this location. Make sure that this driver version is appropriate for your Windows system and Microsoft SQL Server database. Downloads of other JDBC driver versions may also be available:

Download: Microsoft SQL Server JDBC Driver 3.0

- The Microsoft SQL Server JDBC Driver 3.0 is implemented in the file sqljdbc.jar (for Java 1.5) or sqljdbc4.jar (for Java 1.6 and above). You may need to copy the file to the /jar directory of your JSM instance and restart the Java Service Manager before using the driver with Transformation Maps in LANSA Composer.
- If you are using integrated Windows authentication to connect to the Microsoft SQL Server database, then it may also be necessary to copy the appropriate version of the file *sqljdbc\_auth.dll* into your /Windows/System32 directory. Versions of this file for 32-bit and 64-bit Windows environments are supplied and installed with the Microsoft SQL Server JDBC Driver. Refer to the driver documentation for more information.
- Refer to relevant Microsoft resources for more information on the JDBC drivers available for Microsoft SQL Server and the keywords and values that can be specified in the JDBC connection string. Following are some useful references. Note that some of these references relate to specific versions of Microsoft SQL Server and associated software. You should refer to the equivalent pages for the software versions you are using.

Microsoft SQL Server JDBC Driver 3.0

ConnectionStrings.com - Forgot that connection string? Get it here!

## **Example Database Connections for Oracle**

**Note:** Example values and screens given in this section are based on particular database and driver software versions and configurations. They are provided for illustration and guidance. Your own environment may require different values and procedures.

You must be able to establish two types of connectivity to your Oracle database for use with the mapping tool and the maps it generates:

• ODBC connection from the LANSA Composer client computer While defining your map you need to be able to establish an ODBC connection to the Oracle database on your network. This connection is initiated from the client computer running MapForce. Refer to:

Installing and Configuring Oracle Client Software Connecting to Oracle to Define a Transformation Map

• JDBC connection from the LANSA Composer server computer

When the Transformation map runs, it will use a JDBC connection to establish the connection to the Oracle database on your network. This connection is initiated from the server computer on which the transformation map executes and is defined in LANSA Composer by means of a Database Configuration. Refer to:

Connecting to Oracle to Execute a Transformation Map

## **Installing and Configuring Oracle Client Software**

In order to establish an ODBC connection to the Oracle database on your network in the mapping tool, you will need to have the appropriate Oracle ODBC driver and supporting software installed and configured correctly on your client computer.

If you already have the Oracle ODBC client and ODBC driver installed and are able to successfully establish ODBC connections to your Oracle database on the network using another ODBC-aware applications (such as Microsoft Excel, for example), then you can proceed directly to Connecting to Oracle to Define a Transformation Map.

There are several ways to obtain, install and configure the necessary Oracle client and ODBC software. Following are some observations from our own experience during testing with Oracle and creating these examples.

- We downloaded the Oracle Instant Client for Microsoft Windows (32-bit).
- We used version 11.1.0.7.0 after having had unresolved difficulties with earlier versions
- To the extent of our testing, we were able to successfully use this version with our Oracle 10g database.
- It was necessary to separately download and install the "Basic" and "ODBC" packages. The former is a pre-requisite for the latter.
- The installation process was manual. It was necessary to download and unzip files following the instructions provided, read and follow instructions in the associated README files, run the ODBC install script and modify and/or create environment variables. If any step is omitted or incorrectly completed, your ODBC client installation may not operate correctly.
- We encountered a number of issues and noticed other reports on the web. If you have difficulty connecting your Oracle client to your Oracle database on the network, we suggest you search resources on the web, using keywords associated with the symptoms you are experiencing. For issues that we encountered, it was our observation that somebody on the web had encountered similar symptoms and suggested workarounds or fixes are often available, either via Oracle resources or through peer-support networks and forums.

Following are some useful references related to downloading, installing, configuring and trouble-shooting the Oracle Instant Client software:

Oracle Instant Client Downloads

Connecting to Oracle

#### **TNS Service Names**

When you connect to an Oracle database using the Oracle ODBC driver, one of the things you are asked to specify is a TNS Service Name. The TNS Service Name is an Oracle artefact that specifies the information that identifies the server system and database to which you wish to connect.

TNS Service Name definitions are created and maintained using the Oracle SQL\*Net program and are held in a text file named TNSNAMES.ORA. The Oracle client looks for this file in the location specified by the TNS\_ADMIN environment variable.

The Oracle Instant Client "Basic" and "ODBC" packages mentioned above *do not include* the necessary program for maintaining the TNSNAMES.ORA file. You can deal with this in one of the following ways:

- Install and use the full Oracle client instead
- Create the TNS Service Names on a different computer that has the necessary programs and then copy TNSNAMES.ORA to your client computer.
- Manually create and edit the TNSNAMES.ORA file using a plain text editor such as Notepad. To do this, you need to know the format and content of the file. Some of the links provided below describe the format and content of the file and/or provide examples.

In all the above cases, you must ensure that the TNS\_ADMIN environment variable has been set correctly so that the Oracle ODBC driver software can locate the file.

Alternatively, where prompted for a TNS Service Name, you can often enter an SQL Connect URL to specify all the necessary connection information without using TNSNAMES.ORA. This is the approach that has been used in the examples described and shown later in this section. Whether you can use this approach may depend on your specific environment and database configuration.

Following are some references that provide more information on TNS Service Names, their definition and the alternate SQL Connect URL form. There are many more references available on the Internet. Use your Internet search engine to locate further references if required. Tnsnames.ora - Oracle FAQ Local Naming Parameters (tnsnames.ora)

### **Connecting to Oracle to Define a Transformation Map**

While defining a Transformation Map using the mapping tool, you will select the *Database* option from the *Insert* menu (or use the corresponding toolbar button) to insert a database component.



The dialogs you have to complete depend on the type of database connection you insert and on choices you have made in previous sessions in the mapping tool. The following steps describe some of the most common and important dialogs and choices: The actual windows you will see and have to complete in order to establish your database connection will vary according to the type of database, driver, previous database selections and other factors. What follows is an example sequence of windows for establishing a new ODBC connection to an Oracle database running on your network.

1. On the first panel of the *Connection Wizard*, choose Oracle as the database type and click *Next*.

(If the *Connection Wizard* is not shown, click the *Connection Wizard* button to display it)

Select a Database	
Connection Wizard	Connection Wizard
	Please select a source database and click on next. Alternatively you can select a source database and configure the wizard behavior.
Existing Connections	The wizard will have to be configured initially for each database.
(ADD)	C Microsoft Access ( AD0 )
ADO Connections	◯ Microsoft <u>S</u> QL Server ( ADO )
side	
ODBC	C MySQL (ODBC)
ODBC Connections	C IBM <u>D</u> B2 ( ODBC )
	C Sybase ( ODBC )
Global Resources	C PostgreSQL ( ODBC )
	< <u>B</u> ack <u>Next</u> > <u>C</u> lose
Note: These e	xamples will use the ORACLE ODBC driver

**Note:** These examples will use the ORACLE ODBC driver installed with the Oracle Instant Client as described in Installing and Configuring Oracle Client Software.

- 2. When prompted to *Configure: Oracle*, you can do either of the following:
  - If you have already created a Data Source Name (DSN) for your Oracle database, then you can click *Use an existing Data Source Name* and select the DSN from those shown.
  - If you have not yet created a Data Source Name (DSN) for your Oracle database, then you should click *Create a new Data Source Name...* and then select the Oracle ODBC driver from the drop-down list.

The drop-down list may show the *Microsoft ODBC for Oracle* driver, but we recommend using the driver installed with the Oracle client software - its name will vary according to the installation path. (The *Microsoft ODBC for Oracle* driver still requires part or all of the Oracle client software to be present on the client computer, so there seems little to be gained by using this driver.)

If the Oracle ODBC driver is not shown in the drop-down list, then click

the *Edit Drivers* button to add it to the list of drivers that the mapping tool will associate with the Oracle database choice. (When you have done this, you must click the *Back To Configuration Page* button to return to the *Configure: Oracle* window.)

(If the Oracle ODBC driver does not appear in the list of all available drivers, then you may not have run the ODBC install script (odbc\_install.exe) as part of the installation of the "ODBC" package of the Oracle Instant Client)

This example assumes that you are creating a new Data Source Name (DSN).

	Configure: C	Dracle		
onnection Wizard	Select an option how y	you wish to connect to the da	atabase and click N	Vext.
	Create a new Data	Source Name (DSN) with th	e driver:	
Existing	Oracle in instantc	lient11_1	•	
Connections	🔿 Use an existing Da	ta Source Name:		
ADO	O User DSN	O System DSN		Edit Drivers
ADO Connections				
DBC Connections				
Slobal Besources				
1000011000010000	Skip the configurati	on sten for wizard		

3. The Oracle ODBC Driver Configuration window prompts for details for the new Data Source Name (DSN). Complete the details as described below:

Dracle ODBC Driver Confi	guration
Data Source Name	0RA0101 0K
Description	Oracle 10g database (ORA0101) on TEST0101
TNS Service Name	//TEST0101/ORA0101
User ID	system Test Connection
Application Oracle Wo Enable Result Sets	rkarounds   SQLServer Migration   Enable Query Timeout  Read-Only Connection
Batch Autocommit Mode	Commit only if all statements succeed
Numeric Settings	Use Oracle NLS settings

- The first two entry fields (*Data Source Name* and *Description*) provide a name and description for the Data Source Name (DSN). Type a name and description that are appropriate to identify the DSN.
- In the TNS Service Name entry field you can enter either a TNS Service Name (defined in the TNSNAMES.ORA file) or the SQL Connect URL required to identify the server and database to which you wish to connect. The example shows the SQL Connect URL form where TEST0101 is the name of the server hosting the database and ORA0101 is the database/service name. For more information and references refer to TNS Service Names.
- Type the User ID that you will use to connect to your server and database.
- Review and change (if necessary) the options in the tabs at the bottom according to your environment, your database and the requirements of the application.

You can click *Test Connection* to verify that the connection can be established with the details you have entered. When complete, click *OK* to continue. If prompted for a password, type the password associated with the user id that you will use to connect to your server and database.

OK	
Cancel	
About	
_	
	Cancel A <u>b</u> out

4. When your connection has been successfully established, the mapping tool allows you to choose the particular tables you wish to use in your transformation map:

C Source	
💎 🔽 💠   Add/Edit SELECT Statement   Add/Edit Table Relat	ions (0)
ELECT Statements	-

Select the database tables you wish to use and click OK. (If your database tables are not in the schema initially shown, click the arrow next to the schema name and select the required schema from the drop-down list.) A database component representing the selected tables is inserted in the workspace for the map.
## **Connecting to Oracle to Execute a Transformation Map**

When the Transformation map runs, it will use JDBC to establish the connection to the Oracle database on your network. This connection is initiated from the server computer on which the transformation map executes.

LANSA Composer prepares the JDBC connection according to the Database Configuration that you provide as a parameter to the Transformation Map. You must define the Database Configuration with the correct details necessary to establish the particular database connection to be used.

#### **Example Database Configuration**

The following sample values assume that you are using an Oracle JDBC driver to connect to the Oracle database on your network:

Database driver	Following are class names commonly used with the Oracle JDBC drivers:
class	For Oracle 10g and earlier: oracle.jdbc.driver.OracleDriver
	For Oracle 11i and later: oracle.jdbc.OracleDriver
Database connection string	The form of the connection string for the Oracle JDBC driver depends on the driver and the type of connection you wish to make. For example:
	For the JDBC Thin Driver, or Server-side Thin Driver:
	jdbc:oracle:thin:@ <database></database>
	where <database> is either a string of the form //<host>: <port>/<service_name>, or a SQL*net name-value pair, or a TNSEntryName. For example, the following is a sample JDBC connection string to connect to the Oracle database named ORA0101 running on the server named TEST0101:</service_name></port></host></database>
	jdbc:oracle:thin:@//TEST0101/ORA0101
	To make a remote connection with the JDBC OCI (Oracle call interface) driver:
	jdbc:oracle:oci:@ <database></database>

where <database> is either a TNSEntryName or a SQL\*net name-value pair defined in tnsnames.ora.

#### **Further considerations**

- The Database Configuration should specify the username and password of the user authorized to perform the database transactions.
- If necessary, you can download the Oracle JDBC Driver appropriate to your environment from this location:

#### JDBC/UCP Download Page

- The name of the .jar that implements the Oracle JDBC driver depends on the version and features of the JDBC driver that you have downloaded. For example, the base JDBC driver for use with Java 1.5 is ojdbc5.jar. You may need to copy the file to the /jar directory of your JSM instance and restart the Java Service Manager before using the driver with Transformation Maps in LANSA Composer.
- Refer to relevant Oracle resources for more information on the available Oracle JDBC drivers, their installation and configuration requirements and the keywords and values that can be specified in the JDBC connection string. Following are some useful references:

Oracle JDBC Frequently Asked Questions Oracle JDBC FAQ

Top Ten Oracle JDBC Tips | O'Reilly Media

ConnectionStrings.com - Forgot that connection string? Get it here!

## Supported Functionality of the Mapping Tool

This section discusses MapForce features and functionality that are or are not supported for use with LANSA Composer. Please note, however, nothing here adds to Altova's own statements of supported features, functions and platforms for Altova MapForce, nor to LANSA's statements of supported platforms and software. For further information please refer to the Altova documentation and their web-site and to the LANSA web-site.

Supported MapForce Functionality Unsupported MapForce Functionality

# Supported MapForce Functionality

LANSA Composer supports the functionality offered by the mapping tool for mapping between XML, EDI, Excel (OOXML), text (including using FlexText), web service functions and database formats and including the use of constants, filters, If-Else conditions, supplied functions and input functions, provided that:

- the functionality is supported by Altova when generating Java code for the map.
- databases and database drivers used and the operating systems on which they are running are amongst those that are supported by LANSA for use with LANSA software.

Certain properties of a Transformation Map definition are set explicitly by LANSA Composer before generating the Java code for a map. Thus any changes that you make to these properties in your map definition will not affect the generated Java implementation of the map. These include:

- The default output directory for generated code
- The application name
- The base package name for Java generation.
- The output language (LANSA Composer always generates Java code)

# **Unsupported MapForce Functionality**

There are certain functions of Altova MapForce that are explicitly NOT supported for use with LANSA Composer. These are:

- MapForce projects
- Web service projects (publishing web services)
- Any functionality that is not supported by Altova for Java generation
- Maps that use multiple target components or "chained transformations" (such cases can generally be accomplished by making use of intermediate "variable" components or by using more than one map each with a single target component)

Transformation Map definitions that make use of these functions may not integrate correctly with LANSA Composer.

In addition, the following user-interface options for Altova MapForce are not used (or supported) with LANSA Composer:

- MapForce plug-in for MS Visual Studio .NET
- MapForce plug-in for Eclipse

# **Supported Versions of the Mapping Tool**

LANSA Composer version 5.0 is tested and supplied with Altova MapForce version 2014 release 2 (2014R2).

LANSA recommends that you upgrade to the version of Altova MapForce that is shipped with LANSA Composer.

#### Newer Versions of Altova MapForce

From time-to-time, newer versions of Altova MapForce will become available. Usually LANSA will then perform limited testing for the new version of Altova MapForce with the current version of LANSA Composer.

You are strongly advised not to upgrade to a new version of Altova MapForce until LANSA has tested and approved it for use with LANSA Composer.

For the latest information on the versions of Altova MapForce that have been tested with LANSA Composer, refer to:

• Supported Platforms for LANSA Composer

#### **Upgrading Altova MapForce**

Installing the LANSA Composer client software does not automatically upgrade an existing installation of Altova MapForce.

However, the installer for the shipped version of Altova MapForce is supplied on the LANSA Composer installation DVD in the directory WINCLIENT\Installs\ALTOVA. You can run the installer manually to upgrade your version of Altova MapForce.

LANSA recommends that you upgrade to the version of Altova MapForce that is shipped with LANSA Composer. However, before you do so, you should consider the following:

- Altova MapForce may require that your software activation key code signifies a valid and current support and maintenance plan in order to install or upgrade to newer versions. If you have a current software maintenance contract for LANSA Composer with LANSA, then you will normally have received new software activation keys for Altova MapForce when your LANSA Composer software maintenance was renewed..
- LANSA recommends uninstalling the current version of Altova MapForce before installing a newer version.
- It will usually be the case that Transformation Maps saved with a newer version will not be capable of being opened with an older version of Altova

MapForce.

- In order to ensure interoperability across all LANSA Composer clients (with respect to opening, editing and preparing Transformation Maps), LANSA strongly recommends that all LANSA Composer clients use the same version of Altova MapForce.
- It is not normally necessary to re-prepare your Transformation Maps after upgrading to a newer version of Altova MapForce.
- In some cases, users may be using Altova MapForce for applications outside LANSA Composer. In that event, you should consider the effect of upgrading Altova MapForce on those other applications.

#### Appendix D. Commands to Invoke LANSA Composer

LANSA Composer provides interfaces or commands to invoke LANSA Composer functionality on the server and/or the client. The available interfaces or commands are described in this appendix in the following sections:

The COMPOSER command for IBM i servers

Using dxstart.exe to start functions of LANSA Composer on Windows

## The COMPOSER command for IBM i servers

**Note:** The COMPOSER command is provided only for IBM i servers. On Windows servers, LANSA Composer provides alternate means or procedures to accomplish the functions that are provided by the COMPOSER command.

LANSA Composer provides the COMPOSER command for invoking selected LANSA Composer functionality on the LANSA Composer server. Using the COMPOSER command you can:

- Display LANSA Composer system information
- Re-organise the LANSA Composer database
- Run a Processing Sequence
- Apply a IBM i Server License for LANSA Composer
- Start the LANSA Composer tutorial orders application

This appendix provides the following information:

COMPOSER Command Parameters Using the COMPOSER Command COMPOSER Command Examples

#### Also see

Run a Processing Sequence using the COMPOSER command Apply the Server License for LANSA Composer

Required

COMPOSER --- REQUEST ---- ABOUT -----> RUN REORG LICENSE LICENCE TUTORD

Optional

>-- PSEQ ------ processing sequence identifier ----->

>-- PARMS ----- parameter name -- parameter value --> | | <----- 100 max -----

>-- PARTITION -- partition identifier ----->

>-- LANGUAGE --- language code ------|

# **COMPOSER Command Parameters**

## REQUEST

This parameter specifies the LANSA Composer function to be performed. The functions that may be requested are:

- ABOUT: Echos messages containing certain LANSA Composer version and environment information back to the command processor. This information includes LANSA listener, HTTP listener and JSM port numbers.
- RUN: Specifies that a LANSA Composer Processing Sequence, Activity or Transformation Map is to be run. The PSEQ parameter must be specified if REQUEST(RUN) is specified. The PSEQ parameter specifies the identifier of the Processing Sequence, Activity or Transformation Map that is to be run.
- REORG: Specifies that the LANSA Composer database housekeeping task is to be run. The database housekeeping task will use the parameter specifications last used or last saved through the LANSA Composer client. Refer to Database Housekeeping for more information.
- LICENSE or LICENCE: Specifies that a server license for LANSA Composer is to be applied. Refer to Apply the Server License for LANSA Composer for more details.
- TUTORD: Starts the LANSA Composer tutorial orders application.

# PSEQ

For REQUEST(RUN), this parameter must specify the identifier of the Processing Sequence, Activity or Transformation Map that is to be run. You can specify the identifier in one of two ways:

- When running a Processing Sequence, you can specify the external identifier (ID or name) of the Processing Sequence.
- Otherwise you can specify the internal identifier of the Processing Sequence, Activity or Transformation Map. The internal identifier is a unique internal key that is allocated for each item by LANSA Composer when it is created. Its value can be displayed on the *Audit* tab in the LANSA Composer client application.

## PARMS

For REQUEST(RUN), the PARMS parameter may be used to specify parameter values for the Processing Sequence, Activity or Transformation Map that is to

be run. Up to 100 parameter values may be specified.

Parameter values are always provided by parameter name, so you only need to supply values for required parameters or where you wish to use a value different from the default parameter value. You can specify the parameter name and value pairs in any order.

For each parameter for which you wish to provide a value, you must provide:

- *parameter name* the name of the parameter as defined in the Processing Sequence, Activity of Transformation Map definition.
- *parameter value* the value for the parameter.

## PARTITION

This parameter specifies the identifier of the LANSA partition containing the LANSA Composer server installation to be used. If not specified, the COMPOSER command will attempt to determine the partition identifier according to information recorded during the last LANSA Composer installation into this system.

## LANGUAGE

This parameter specifies the national language to be used. If not specified, the language defaults to the partition default language.

## Using the COMPOSER Command

The following sections describe some further considerations for using the COMPOSER command:

Invoking the COMPOSER Command Submitting the COMPOSER Command to Batch Including the COMPOSER Command in Your Own Programs Monitoring Messages Issued by the COMPOSER Command

# **Invoking the COMPOSER Command**

The COMPOSER command is installed in the LANSA program library of the LANSA system in which LANSA Composer is installed. In a default LANSA Composer installation this is LICPGMLIB, but it may be different on your system.

The COMPOSER command can be entered at a command line display by typing the command and pressing F4 to prompt for the command parameter values:

#### COMPOSER

If the LANSA program library is not already in the library list, you can enter the command as:

#### <pgmlib>/COMPOSER

where <pgmlib> is the name of the program library for the LANSA system in which LANSA Composer is installed. For example, if you installed LANSA Composer into the default location, you would type:

#### LICPGMLIB/COMPOSER

Once invoked, the COMPOSER command will automatically establish the environment it needs to operate, including modifying the library list if necessary. However, it is your responsibility to ensure that the library list includes any necessary application libraries.

#### Submitting the COMPOSER Command to Batch

When using the COMPOSER command to run a Processing Sequence, you can submit the command to batch. For example, you could use a command like this to submit the supplied processing sequence TUT\_01\_P1 to run in batch :

SBMJOB JOB(TUT\_01\_P1) CMD(<pgmlib>/COMPOSER REQUEST(RUN) PSEQ(TUT\_01\_P1) PARMS((YOURINITIALS 'FS') (YOURMESSAGE 'Hello World') (YOURNAME 'Fred Smith')))

(where <pgmlib> is the name of the program library for the LANSA system in which LANSA Composer is installed).

# **Including the COMPOSER Command in Your Own Programs**

When using the COMPOSER command to run a Processing Sequence, you can compile it into your own 3GL programs, typically i5/OS Control Language (CL) programs.

In this way you can deeply integrate your LANSA Composer solutions within your own application processing and increase the flexibility of your solutions by passing variable data supplied by your application into the Processing Sequence using parameters.

**Note:** You should not attempt to directly invoke the COMPOSER command from a LANSA application, although you can submit it to batch. Refer to Appendix E. Using LANSA Composer with LANSA Applications for more information.

## **Monitoring Messages Issued by the COMPOSER Command**

When the COMPOSER command is used to run a Processing Sequence, it will issue message CPF9898 as an \*ESCAPE message if the Processing Sequence does not end normally. You can monitor for CPF9898 in your CL programs using the MONMSG CL command.

# **COMPOSER Command Examples**

Example 1: Display LANSA Composer server version and environment information:

#### <pgmlib>/COMPOSER REQUEST(ABOUT)

where **<pgmlib>** is the name of the program library for the LANSA system in which LANSA Composer is installed. For example, if you installed LANSA Composer into the default location, you would type:

#### LICPGMLIB/COMPOSER REQUEST(ABOUT)

If you typed the above command at a command line, LANSA Composer will display messages containing the LANSA Composer version and environemt information. For example::

#### LANSA Composer Server version 3.0 (Build 307) LANSA V11SP5 IBM i server 6.1

Example 2: Initiate applying a server license to the LANSA Composer server system:

#### <pgmlib>/COMPOSER REQUEST(LICENSE)

where **<pgmlib>** is the name of the program library for the LANSA system in which LANSA Composer is installed. For example, if you installed LANSA Composer into the default location, you would type:

#### LICPGMLIB/COMPOSER REQUEST(LICENSE)

Example 3: Run the supplied processing sequence TUT\_01\_P1 :

<pgmlib>/COMPOSER REQUEST(RUN)
pseq(TUT\_01\_P1)
parms( (YOURINITIALS 'FS')
(YOURMESSAGE 'Hello World')
(YOURNAME 'Fred Smith') )

Example 4: Run the supplied activity FTP\_INBOUND using the EXAMPLE\_FTPIN configuration :

<pgmlib>/COMPOSER REQUEST(RUN)
pseq(4B3DEDA9F50B4C53885BE5318F12DDCA)
parms( (FTPCONFIG 'EXAMPLE\_FTPIN') )

# Using dxstart.exe to start functions of LANSA Composer on Windows

LANSA Composer provides dxstart.exe for starting functionality of the LANSA Composer client and/or of the LANSA Composer Windows server. It is installed in the X\_WIN95\X\_LANSA\Execute directory of the LANSA Composer client or server installation. Using dxstart.exe you can:

- Start the LANSA Composer client application
- Start the LANSA Composer Document Manager application
- Execute a processing sequence run "shortcut" from the client computer
- Re-organise the LANSA Composer database

When you install LANSA Composer, shortcuts are automatically created for the first two of these functions. If you need to, you can use the information contained here to create your own shortcuts or integrate dxstart.exe into your desktop or server environment in other ways that you choose. This appendix provides the following information:

dxstart.exe Syntax and Parameters dxstart.exe Configuration File dxstart.exe Examples

## dxstart.exe Syntax and Parameters

You can specify switches and parameters on the dxstart.exe command line to control its behaviour.

```
dxstart.exe /request={dxclient|dxtxmon|dxrun|dxreorg|dxregister}
```

/lang=lll /part=ppp /offline /shortcut=path

# /<mark>request= {dxclient|d</mark>xtxmon|dxrun|dxreorg|dxregister}

This switch specifies the LANSA Composer client or server function to be performed. The functions that may be requested are:

- dxclient: Starts the LANSA Composer client application. This is the default if the /request switch is not provided or the value is not recognised.
- dxtxmon: Starts the LANSA Composer Document Manager application.
- dxrun: Executes a processing sequence run "shortcut" file. The path to the shortcut file must be provided using the /shortcut switch. In normal usage, LANSA Composer registers the file extension associated with a LANSA Composer processing sequence run "shortcut" file (.dxrun) such that the default Windows Explorer action for files with that extension is to invoke dxstart.exe with this switch and the /shortcut switch.
- dxreorg: Executes the LANSA Composer database housekeeping task. The database housekeeping task will use the parameter specifications last used or last saved through the LANSA Composer client. Refer to Database Housekeeping for more information. This function can only be executed on the LANSA Composer Windows server.
- dxregister: May be used to explicitly invoke dxtstart.exe to register the .dxrun file with Windows Explorer and associate dxstart.exe with it. Normally this occurs in the course of standard LANSA Composer installation and processing.

## /lang=lll

This switch may be used to specify the national language used to start LANSA Composer. If not specified, LANSA Composer will automatically detect French and Japanese systems and start the LANSA Composer client in the appropriate language. If this switch is specified, its value overrides the automatic detection and any value that may be specified in the dxstart.cfg configuration file.

## /part=ppp

This switch may be used to specify the partition name used to start LANSA Composer. If specified, this value overrides any value that may be specified in the dxstart.cfg configuration file. In a standard LANSA Composer installation it should not be necessary to specify this switch as the software is always installed in partition named LIC. The partition specified here does not affect or override the partition name used for the connection to the LANSA Composer server.

## /offline

For /request=dxclient or /request=dxtxmon, this switch may be specified to cause the LANSA Composer client or LANSA Composer Document Manager application to start in offline mode, without showing the *Connect to LANSA Composer Server* dialog.

You would never use this switch in a standard LANSA Composer client installation, as such an installation does not have a local LANSA Composer database and therefore cannot run in offline mode.

This switch is typically used with the LANSA Composer client that is installed with a LANSA Composer Server for Windows installation.

## /shortcut=path

For /request=dxrun, this switch specifies the full path to the LANSA Composer processing sequence "shortcut" file to be executed. Such a file usually has a file extension of .dxrun. If the path contains spaces or other special characters it should be fully-enclosed in double-quote marks.

# dxstart.exe Configuration File

In a standard LANSA Composer installation, dxstart.exe requires only the information described above to successfully start the requested function.

In exceptional circumstances, it may be necessary to override aspects of dxstart.exe's default behavior. This can be done by altering the contents of the dxstart.cfg file contained in the same folder as dxstart.exe. The presence of this file is optional - if it is not present, dxstart.exe will use default behaviours. The following is an example dxstart.cfg file. All the configuration parameters

in this example are commented by the preceding semi-colon (;) and thus are not effective. The dxstart.exe program will exhibit default behavior in this case.

```
_____
; THIS file can be used to configure dxstart.exe so that it can locate
; and correctly execute the LANSA and LANSA Composer executable files.
; dxstart.exe is used to:
; - start the LANSA Composer client
 (dxstart.exe /request=dxclient)
; - start the LANSA Composer Document Manager
 (dxstart.exe /request=dxtxmon)
; - run client-side LANSA Composer processing sequence run "shortcuts"
 (dxstart.exe /request=run /shortcut=%1)
 (normally accomplished by Windows shell file association for .dxrun)
; - execute the LANSA Composer database housekeeping task
 (dxstart.exe /request=dxreorg)
; In a standard LANSA Composer installation, the defaults used by
; dxstart.exe should suffice - it should NOT be necessary to enable
; ANY settings in this file.
; For a LANSA Composer client that is or may be used in OFFLINE mode,
; it will be necessary that this file is configured correctly to permit
; the offline LANSA Composer client to access the local database.
```

; Typically, this applies to:

; - an offline client used on and with a LANSA Composer Windows server ; installation,

; - a LANSA Composer development environment (applies to LANSA LPC only)

; Lines beginning with ; are comments and will NOT be effective.

; -----

[dxStart] ;x\_win95=C:\Program Files\LANSA Composer\X\_WIN95

[x\_run]
;user=LICPGMLIB
;part=LIC
;lang=ENG
;dbid=LX\_LANSA
;dbut=SQLANYWHERE
;dbus=DBA
;pswd=SQL
;extra=DBID=LX\_LANSA DBUT=SQLANYWHERE DBUS=DBA
PSWD=SQL

#### dxstart.exe Examples

Example 1: Start the LANSA Composer client application: dxstart.exe

Example 2: Start the LANSA Composer Document Manager application in French:

```
dxstart.exe /request=dxtxmon /lang=FRA
```

Example 3: Initiate a LANSA Composer processing sequence run from the client computer using the processing sequence id and parameter values specified in the file c:\example.dxrun:

```
dxstart.exe /request=dxrun /shortcut="c\example.dxrun"
```

Example 4: Execute the LANSA Composer database housekeeping task on a Windows server:

dxstart.exe /request=dxreorg

# Appendix E. Using LANSA Composer with LANSA Applications

LANSA Composer is an application that is functionally independent of other LANSA products. It is not necessary at all that you use or have installed other LANSA products (including LANSA development tools) in order to make use of LANSA Composer.

If, however, you do use LANSA development tools to build your own applications, two factors position LANSA Composer to provide the best possible integration with those LANSA applications:

1. LANSA Composer is built by LANSA

2. LANSA Composer is itself built using LANSA development tools

Typical integration between LANSA Composer and a LANSA application will take place in the course of executing a LANSA Composer Processing Sequence.

The integration may occur in either or both of two directions:

LANSA Application Initiates a Processing Sequence Processing Sequence Invokes Functions or Components of a LANSA Application

There are several standard means that LANSA Composer supports for a LANSA application to integrate with and extend your business process integration solutions. These include:

• The CALL\_FUNCTION activity

LANSA Composer supplies the ready-to-use CALL\_FUNCTION activity for calling existing LANSA functions. The LANSA functions to be called may exist in the same or a different LANSA system and/or partition as LANSA Composer.

• Activity processors for custom activities

In fact, the processors for custom activities that you define must be developed using Visual LANSA, even if they simply exist to call existing 3GL programs. (Your LANSA Composer supplier can usually do this for you if LANSA development tools and/or LANSA development skills are not available to you.)

• User-defined event notification handlers

As an alternative to the built-in email and SMS event notification handlers, it

is possible to define your own event notification handler as a LANSA function.

• LANSA applications can initiate LANSA Composer Processing Sequences

There are several ways to initiate a Processing Sequence, most of which can be used by any application. However, from LANSA V12SP1, specific builtin functions are provided with LANSA specifically for use by LANSA applications to initiate LANSA Composer Processing Sequence.

Also refer to:

Required Characteristics of the Common LANSA environment

LANSA application developers should be guided in all cases by the relevant LANSA documentation.

# LANSA Application Initiates a Processing Sequence

There are several ways that an application can initiate a Processing Sequence. They are fully described in Run a Processing Sequence.

However, for a LANSA application, there are some particular and additional considerations.

The recommended way for a LANSA application to directly execute a Processing Sequence is by means of the built-in functions supplied with LANSA V12SP1 and above. Refer to Run a Processing Sequence from a LANSA application for more information. The considerations below do not apply if you use the supplied built-in functions in your LANSA application.

If your LANSA application will submit the Processing Sequence such that it executes in a separate process or job, then you are free to use any of the described means to do so, even when the LANSA application is in a different LANSA system and/or partition to LANSA Composer.

If, however, you wish your LANSA application to directly execute the Processing Sequence (other than by means of the built-in functions provided), then you should consider the following:

- You must not invoke the Processing Sequence using any method that will cause one of the LANSA entry points (LANSA command or program, X\_RUN) to be executed, as this will lead to recursion issues. In particular, this means:
  - You may not invoke the COMPOSER command
  - You may not invoke the DXP1RUN function via the LANSA command or by executing X\_RUN
- The LANSA application code unit (function or component) that will invoke the Processing Sequence must reside in the same LANSA system and partition as LANSA Composer.

These considerations apply whether the invocation is direct or indirect. In other words, you cannot circumvent these constraints by initiating the Processing Sequence from a non-LANSA program that you invoke from your LANSA application.

## **Processing Sequence Invokes Functions or Components of a** LANSA Application

LANSA Composer may be extended in a number of ways that involve a Processing Sequence invoking customer or 3rd-party defined functions or components. These include:

• The CALL\_FUNCTION activity

LANSA Composer supplies the ready-to-use CALL\_FUNCTION activity for calling existing LANSA functions. The LANSA functions to be called may exist in the same or a different LANSA system and/or partition as LANSA Composer.

• Activity processors for custom activities

In fact, the processors for custom activities that you define must be developed using Visual LANSA, even if they simply exist to call existing 3GL programs. (Your LANSA Composer supplier can usually do this for you if LANSA development tools and/or LANSA development skills are not available to you.)

• User-defined event notification handlers

As an alternative to the built-in email and SMS event notification handlers, it is possible to define your own event notification handler as a LANSA function.

When a Processing Sequence invokes customer or 3rd-party defined LANSA functions or components in any of these ways there are some particular and additional considerations. When designing such a business process integration solution you should consider the following:

- Except for functions called via the CALL\_FUNCTION activity, the LANSA application code unit (function or component) to be invoked must reside in the same LANSA system and partition as LANSA Composer.
- You must not invoke the LANSA application code unit using any method that will cause one of the LANSA entry points (LANSA command or program, X\_RUN) to be executed, as this will lead to recursion issues.

These considerations apply whether the invocation is direct or indirect. In other words, you cannot circumvent these constraints by invoking a non-compliant

LANSA application code unit from another LANSA application code unit that is compliant. Nor can you do so by means of an intermediate non-LANSA program that you invoke from LANSA Composer or from your compliant LANSA application code unit.

## **Required Characteristics of the Common LANSA environment**

NOTE: This information is provided primarily for the benefit of preexisting LANSA Composer installations that may have used this type of installation. From version 4.0 of LANSA Composer, it is no longer possible or necessary to create new installations of LANSA Composer in an existing LANSA system that is used by other LANSA applications.

As described in preceding sections, some of the possible forms of integration between LANSA Composer and a LANSA application require that LANSA Composer and the LANSA application code unit reside in the same LANSA system and partition.

In those cases, therefore, the LANSA system and partition must meet the minimum requirements for LANSA Composer. Consequently, the LANSA application code unit must also be compatible with those minimum requirements. In particular, the common LANSA partition:

- Must be RDMLX-enabled
- Must be web-enabled
- Must be multilingual-enabled
- Must be at least at the LANSA version and EPC level required by the version of LANSA Composer that is installed

## Appendix F. The LANSA Composer Request Server

The LANSA Composer Request Server provides support for the tightest possible integration between LANSA Composer and other LANSA applications, permitting each to invoke functions in the other, even though they may be executing on different server systems, in different LANSA run-time systems and/or different LANSA partitions, and even if they are running in different versions or EPC levels of the LANSA run-time.

In particular, the LANSA Composer Request Server supports the following:

• LANSA Composer "calling" LANSA

Using the CALL\_FUNCTION activity in a LANSA Composer Processing Sequence to "call" a LANSA function on the same or a different server system and/or in a different LANSA system and/or partition and exchange variable information with it.

• LANSA Composer "calling" LANSA Composer

Using the COMPOSER\_RUN activity in a LANSA Composer Processing Sequence to run a Processing Sequence in another installation of LANSA Composer on the same or a different server system and/or in a different LANSA system and/or partition and exchange variable information with it.

• LANSA "calling" LANSA Composer

Using the COMPOSER\_RUN built-in function in a LANSA application to directly "call" a LANSA Composer Processing Sequence on the same server system in a different LANSA system and/or partition and pass variable information to be used as Processing Sequence parameter values.

By and large, the operation of the LANSA Composer Request Server is transparent to users of the system. The LANSA Composer designer or the LANSA application developer simply needs to provide appropriate parameter values to the CALL\_FUNCTION activity or the COMPOSER\_RUN built-in function. If needed, the request will be directed through the LANSA Composer Request Server automatically.

The LANSA Composer Request server works by executing the request in a separate process or job. This occurs transparently and is managed by the request server to ensure that no conflict is possible arising from attempting to simultaneously use two LANSA run-time systems.

The implementation of the LANSA Composer Request Server is specific to the server operating environment being used. In other words, LANSA Composer provides two implementations:

- LANSA Composer Request Server for IBM i servers
- LANSA Composer Request Server for Windows servers

For the most part, the LANSA Composer designer or the LANSA application developer need not be concerned with any differences between the implementations for IBM i and Windows servers. However, there are some differences in the constraints and behaviors that may be relevant or important in some cases. Amongst other things, this appendix documents such differences. In addition, there are some things you need to understand about the execution context for requests that are processed through the LANSA Composer Request Server. Furthermore, on IBM i servers, you may need to manually configure the LANSA Composer Request Server for optimal operation in your operating environment.

The following topics provide additional information:

What You Need to Know About Requests Processed Through the LANSA Composer Request Server

How the LANSA Composer Request Server for IBM i Works

How the LANSA Composer Request Server for Windows Servers Works

Connecting to the LANSA Composer Request Server on a Remote System

LANSA Built-in Functions that use the LANSA Composer Request Server

LANSA Programming Considerations for Called Functions

Configuring the LANSA Composer Request Server for IBM i to Suit Your Needs

# What You Need to Know About Requests Processed Through the LANSA Composer Request Server

There are several key things to understand about requests that are processed through the LANSA Composer Request Server. Some of these have consequences that may have a significant bearing on the way that you design your solutions that use the LANSA Composer Request Server.

1. The function "call" happens in a different process or job or even on a different computer

Normally it is impossible for a LANSA application running in one LANSA system and/or partition to directly call a LANSA function running in a different LANSA system and/or partition. In order to emulate this capability, the LANSA Composer Request Server directs the requests to a separate process or job or even a separate server system, exchanging the request and response data with that job as required.

In many respects, this fact is transparent to the requesting application. Depending on your application design it may have no functional implications at all. However, it does mean that the execution context for the function call may not be what you might otherwise expect it to be.

Understanding this point is fundamental - the remaining points in this section are largely a direct consequence of this fact.

2. On IBM i servers, the function "call" executes with the user credentials of the request server job

The LANSA Composer Request Server for IBM i makes no attempt to exchange user credentials with the request server job that executes the request and the request server does not switch user contexts. The request that executes in the request server job will execute with the user credentials of the request server job. If necessary, you can change the way that the request server jobs are started such that they use a different user profile of your choosing. Refer to Configuring the LANSA Composer Request Server for IBM i to Suit Your Needs for further information.

3. The function "call" happens in the execution environment of the request server job

Because the requests execute in a separate process or job, it follows that they are subject to the execution environment of that request server job - not of the job that made the request. Particularly on IBM i servers, this is

significant for a range of job attributes, including (but not limited to):

- File overrides in the requesting job do not apply to the request
- Open data paths in the requesting job are not shared with the request
- The contents of the QTEMP library of the requesting job are not available to the request
- The request executes with the library list of the request server job (not the requesting job)
- 4. The execution environment of the request server job does not persist from one call to another

It is not valid to assume that the execution environment of the request server job(s) that process your requests will persist from one call to another. For example, if a called LANSA function on an IBM i server makes changes to the library list or creates objects in QTEMP, you must not design your solution relying on the fact that those changes will still be in effect for the next called function. There are two main reasons for this:

• It may not be the same process or job

On Windows servers, every request directed through the LANSA Composer Request Server executes in a new process.

On IBM i servers, to handle a given load, more than one request server job may be active to service requests to a particular LANSA system. There is no guarantee, for example, that the same job will execute the requests for successive CALL\_FUNCTION activities in a LANSA Composer Processing Sequence.

• Each request server job reinitializes its state between requests

On IBM i servers, in order to ensure the integrity of the state of the LANSA execution environment from one request to another, the request server jobs perform certain re-initialisation between requests.

## How the LANSA Composer Request Server for IBM i Works

The diagram below provides an overview of the way the LANSA Composer Request Server for IBM i works for the case where a LANSA Composer Processing Sequence uses the CALL\_FUNCTION activity to synchronously "call" a LANSA function that may be in a different LANSA system and/or partition. The numbers in the diagram refer to the notes that follow.

Although this diagram illustrates the CALL\_FUNCTION activity, the processing is similar for the reverse case where a LANSA application "calls" a LANSA Composer Processing Sequence using the COMPOSER\_RUN built-in function.



- 1. The activity processor for the CALL\_FUNCTION activity posts the request to the request server by adding one or more entries to header and detail data queues used by the LANSA Composer Request Server. It may also start the request server job automatically if it detects that there is not one available.
- 2. The LANSA Composer Request Server receives details about the request from the data queues.

- 3. For the CALL\_FUNCTION case, the request server populates the exchange list in the target LANSA system with any exchanged variables and calls the specified LANSA function. It then receives exchanged variable values from the LANSA exchange list.
- 4. The request server posts any response data to the response data queue.
- 5. The activity processor for the CALL\_FUNCTION activity receives the response data from the data queue, determines the result and updates values in the Processing Sequence variable pool as necessary.
#### How the LANSA Composer Request Server for Windows Servers Works

The implementation of the LANSA Composer Request Server for Windows servers is simpler than the IBM i implementation. There are no data queues and no subsystems to configure.

On a Windows server, each request to the LANSA Composer request server spawns a new WIndows process that executes the request and communicates with the requesting process. Communications between the processes conveying the exchange variables and results and any necessary configuration information is accomplished using Windows messages. When a request completes, the process that executed it ends too.

There are no additional configuration tasks necessary for the LANSA Composer Request Server for Windows servers.

## **Connecting to the LANSA Composer Request Server on a Remote System**

There are two activities supplied with LANSA Composer that make use of the LANSA Composer Request Server. They are:

CALL\_FUNCTION and

COMPOSER\_RUN

In both cases, an activity parameter is used to specify the name of a LANSA system configuration that contains the details necessary to connect to another LANSA system to run a LANSA function or a LANSA Composer processing sequence in that system.

In the simplest case, the other LANSA (or LANSA Composer) system resides on the same server as the requesting LANSA Composer system.

But it is also possible to define the LANSA system configuration such that the activities can call a LANSA function or execute a LANSA Composer Processing Sequence on a different (\*) server system.

\* Note a LANSA Composer Remote Request Server license is required for the remote server in order to call a LANSA function or run a Processing Sequence on a different server system.

NB: At time of writing (with reference to LANSA V12SP1), connection from one IBM i server to another IBM i server is not supported. This restriction may be lifted in future LANSA versions.

In addition to correctly defining the connection in the LANSA system configuration, there are some other pre-requisites to successfully using this support. These include:

LANSA Composer must be installed and licensed on the remote system

The LANSA Listener must be active in the remote LANSA Composer system

The requesting system must contain appropriate LANSA communications routing records for the remote system

## LANSA Composer must be installed and licensed on the remote system

You must install LANSA Composer on the remote system in order to provide

the remote request server components that allow the communications link to be completed.

This installation of LANSA Composer might not be used for any other purpose than to provide those remote request server components. It does not need to be a fully-licensed LANSA Composer system unless you wish to design or execute Processing Sequences on the remote system However, a LANSA Composer Remote Request Server license is required for the remote server.

### The LANSA Listener must be active in the remote LANSA Composer system

The requesting LANSA Composer system will connect to the remote system through the LANSA listener in the remote LANSA Composer system. Therefore the LANSA listener must be active in that system in order that it may receive and process the requests.

# The requesting system must contain appropriate LANSA communications routing records for the remote system

A vital part of the LANSA system configuration is the *Server LU partner name* that specifies the logical name that refers to the LANSA listener on the remote system.

The name specified must identify a LANSA communications routing record that specifies, inter alia, the network name or address of the remote system and the TCP port number on which the remote LANSA Composer system is listening.

- When the requesting LANSA Composer system is running on a Windows server, you create and/or maintain these records using the *LANSA Communications Administrator*.
- When the requesting LANSA Composer system is running on an IBM i server, you create and/or maintain these records using LANSA's *Work with Communications Routing Records* screen.

NOTE: the LANSA built-in functions used for LANSA development activity to integrate with LANSA Composer do not support connecting to a request server on a remote system. They can only be used for integrating LANSA applications and LANSA Composer systems that execute on the same server system.

### LANSA Built-in Functions that use the LANSA Composer Request Server

If your organization uses LANSA development tools, you should know that as of LANSA V12SP1, a set of built-in functions is available in the LANSA development environment that permit your LANSA applications to interact with LANSA Composer through the LANSA Composer Request Server.

The built-in functions are:

COMPOSER\_USE : Specifies connection information for subsequent requests

COMPOSER\_RUN : Runs a LANSA Composer Processing Sequence

COMPOSER\_CALLF : Calls a LANSA function

The COMPOSER\_RUN built-in function is for use in a LANSA application to execute a LANSA Composer Processing Sequence through the LANSA Composer Request Server.

The COMPOSER\_CALLF built-in function is intended to be used in a custom activity processor that executes in the LANSA Composer environment. It is to permit your custom activity processor to call a function in your LANSA application through the LANSA Composer Request Server.

Refer to the LANSA Technical Reference for detailed programming instructions for these built-in functions.

NOTE: the LANSA built-in functions do not support connecting to a request server on a remote system. They can only be used for integrating LANSA applications and LANSA Composer systems that execute on the same server system.

### **LANSA Programming Considerations for Called Functions**

When developing LANSA functions that will be called from a LANSA Composer Processing Sequence using the CALL\_FUNCTION activity via the LANSA Composer Request Server, some particular LANSA programming considerations apply:

- Either RDML functions or fully RDMLX-enabled functions may be called.
- The partition containing the function to be called must be a multilingual partition.
- If \*DIRECT is specified or assumed for the PROCESS parameter of the CALL\_FUNCTION activity, the function must be defined with FUNCTION OPTIONS(\*DIRECT)

If you intend to exchange variable data with the called function using the EXCHnn parameters of the CALL\_FUNCTION activity, these additional considerations apply:

- For functions that will be called on IBM i servers, position 487 of LANSA data area DC@A01 must be set to 'Y' (in the LANSA system containing the LANSA function to be called) before compiling and executing the function. If this condition is not met, the called function will not correctly receive or return the EXCHnn variable values.
- The function must define the fields EXCH01 through to EXCH07 in order to receive values (via the exchange list) that are specified in the corresponding CALL\_FUNCTION activity parameters.
- The function must use the EXCHANGE command with fields EXCH01 through to EXCH07 in order to return values (via the exchange list) for variables that are specified in the corresponding CALL\_FUNCTION activity parameters.

Refer to LANSA documentation for more information on LANSA features referred to above.

### **Configuring the LANSA Composer Request Server for IBM i to Suit Your Needs**

This section applies only to the LANSA Composer Request Server for IBM i. On Windows servers, every request processed through the LANSA Composer Request Server executes in a separate new process, and the number of request server requests that can be simultaneously processed is limited only by system memory and processing capacity. No additional configuration is necessary for the LANSA Composer Request Server on a Windows server.

On many systems, the LANSA Composer Request Server for IBM i servers will work correctly and provide adequate adaptive load-balancing as installed.

To function correctly, the LANSA Composer Request Server for IBM i must be capable of submitting its request server jobs to a job queue and subsystem that are active and capable of executing multiple active jobs simultaneously.

As installed the LANSA Composer Request Server for IBM i will do this automatically using job queue QUSRNOMAX. This job queue is supplied by IBM with the i5/OS operating system and is usually associated with subsystem QUSRWRK.

Other features of the installed behavior of the LANSA Composer Request Server for IBM i include:

- Request server jobs are automatically submitted on demand that is, when a requesting job makes a request that needs the LANSA Composer Request Server for IBM i.
- The LANSA Composer Request Server for IBM i will activate up to a maximum of five request server jobs at any one time (for each target LANSA system), to cope with possible load from multiple requestor jobs.
- The request server jobs will self-terminate after five minutes of inactivity.

Together these features provide a server configuration that is capable of adapting to load spikes, but is unobtrusive for the relatively modest level of usage that it will serve in most environments.

Nevertheless, you may need or want to adapt the installed configuration for your own use. All the foregoing features and values may be altered to suit your environment. Such adaptations might include:

• Using a different job queue and/or subsystem for the request server jobs

- Changing the maximum request server jobs from the supplied default value of five, or changing other values that affect when and how the LANSA Composer Request Server for IBM i will submit request server jobs.
- Taking full control yourself of the submission of request server jobs by creating your own work management configuration (subsystem descriptions, job descriptions, autostart job entries etc).

The following topics provide additional information to allow you to configure the LANSA Composer Request Server for IBM i to suit your needs:

Library List Considerations for the Request Server Jobs LANSA Composer Request Server for IBM i Configuration Settings

### Library List Considerations for the Request Server Jobs

The prime purpose of the request server jobs is to call a LANSA function, often on a different server system and/or in a different LANSA system and/or partition to the requesting job.

The LANSA Composer Request Server for IBM i does this using a LANSA program library name and a partition name passed to it as part of the request. The request server job uses this information to invoke the requested function in LANSA, relying entirely on LANSA's standard library list management features - in other words, LANSA will alter the library list to contain the libraries necessary for execution of functions in the specified LANSA system and partition, according to the applicable LANSA system and partition settings.

The initial library list for the request server job should never contain any LANSA libraries. In particular, if the initial library list contains LANSA libraries for a LANSA system other than that in which the request is executing, severe and unpredictable execution errors may occur.

Note that the library containing the LANSA Composer Request Server for IBM i software does NOT need to be in the library list, providing the request server job is initiated using a qualified call to the DXRQSSERV program. The request server, although it may be installed in a LANSA system or partition library, is entirely composed of non-LANSA software, does not itself use the LANSA run-time (other than to execute the specified request) and makes fully-qualified references to its own components using the library name from which the DXRQSSERV program is started.

If your application requires libraries in the library list other than those that are automatically added by LANSA, then you will have to make provision yourself for adding those libraries. This would typically be done in one of two ways, depending on your specific requirements:

- By specifying the libraries in the initial library list for all request server jobs;
- By having your called function alter the library list as required.

# LANSA Composer Request Server for IBM i Configuration Settings

Settings that affect the behavior of the LANSA Composer Request Server for IBM i are contained in the data area DXRQSSERV in the library containing the LANSA Composer Request Server software. You can alter these settings in order to configure the LANSA Composer Request Server to suit your requirements and your operating environment. To alter the settings, use the CHGDTAARA (Change Data Area) command.

Note that the settings are all accessed by their position in the data area. Any text labels contained in the data area are present for ease-of-reference and are NOT used by the software to locate or identify the settings.

Display Data Area			
Data area Library Type Length . Text	: DXRQSSERV : LICLICLIB : *CHAR : 2000 : LANSA Composer request se	System: rver	LANSA01
	Value		
Offset 0 50	*+1+2+3+4. 'WORKMANAGEMENT:*AUTO 'MAYMON:00005 WAITMON:00003 INACTIVE:00300	+5	
100	1 111/100.00003 WHITHOU.00003 INHEIIVE.00000		
150			
200			
250			
300			
350 400			
Press Enter to continue.			More
F3=Exit	F12=Cancel		

### Work Management Mode (positions 16 - 25)

The work management mode determines whether the LANSA Composer Request Server automatically starts the request server jobs on demand and according to other settings described herein.

If the value is \*AUTO, then the LANSA Composer Request Server will automatically start the request server jobs on demand.

If any other value is specified, then the LANSA Composer Request Server will NOT automatically start the request server jobs on demand. In this case it is

your responsibility to start the request server jobs.

### Maximum Request Server Jobs (positions 58 - 62)

When the work management mode is \*AUTO, this setting determines the maximum number of request server jobs that the LANSA Composer Request Server will activate at any one time for each target LANSA system. If no request server job is "available" to process a request (within the wait time specified by the next setting), the LANSA Composer Request Server will check how many request server jobs are already active for a given target LANSA system. If the number is less than specified by this setting, the LANSA Composer Request Server will start another request server job to satisfy the request. Otherwise, no new job is started and the request is simply posted for an active request server job to process when one becomes available. In the latter case, the request may timeout or expire (according to parameters specified for the request) if an active request server job does not become available in time to process the request.

If the work management mode is not \*AUTO, this setting is not used.

### Wait Time for an Available Request Server Job (positions 72 - 76)

When the work management mode is \*AUTO, this setting specifies the number of seconds the LANSA Composer Request Server will wait for a response from an available request server job before starting a new request server job (subject to the maximum specified by the previous setting) to satisfy the request. If the work management mode is not \*AUTO, this setting is not used.

# Maximum Inactive Time for Request Server Job (positions 87 - 91)

This settings specifies the maximum number of seconds a request server job will remain active while waiting for new requests. When the specified number of seconds has elapsed since the last completed request processed by the job, the request server job will automatically end.

This setting applies irrespective of whether \*AUTO is specified for the work management mode settings. However, if the work management mode is not \*AUTO, you will probably need to increase the value specified in this setting to a very large value or make provision in your work management arrangements for new request server jobs to be periodically started.

### Skeleton SBMJOB (Submit Job) Command (positions 769 - 1024)

When the work management mode is \*AUTO, this setting specifies the

command string used by the LANSA Composer Request Server to start new request server jobs. For example, as shipped, the following command string is used:

SBMJOB CMD(CALL &MONLIB/DXRQSSERV PARM('&PGMLIB')) JOB(LCRQSSERV) JOBQ(QUSRNOMAX) CURLIB(\*USRPRF) INLLIBL(\*SYSVAL) USER(&PGMLIB)

If you wish to customise the way new request server jobs are started, you may alter this command string as required. If you do so, please be guided by the following:

• The request server job must run program DXRQSSERV in the library specified by the placeholder variable &MONLIB and with a single parameter that specifies the LANSA program library name for which it will service requests.

If you wish, you could customise the command string to submit a call to your own custom program as long as:

a)Your program calls DXRQSSERV as described above.

b)Your program is a non-LANSA program and does not invoke any LANSA program.

- Most typically (and certainly whenever synchronous request are used), the request server job should be capable of becoming active immediately. That may mean, for example, that you should use a job queue that is connected to a subsystem that is active and that is capable of simultaneously executing more than one active job.
- You need to be careful about the current library and the initial library list used. Specifically, it should not include any LANSA libraries. Refer to Library List Considerations for the Request Server Jobs for more information.
- The command string can and should use the following placeholder variables to refer to the corresponding library names. LANSA Composer Request Server will replace all instances of the placeholder variables with the corresponding library names, adjusting the command length as necessary to accommodate them.

&MONLIB : refers to the library containing the LANSA Composer Request Server software.

&PGMLIB : refers to the LANSA program library for which this request server job will execute requests.

• The command string (after replacement of the placeholder variables) will be executed via the IBM-supplied QCMDEXC program.

If the work management mode is not \*AUTO, this setting is not used.

## Appendix G. National Language and Multilingual Support in LANSA Composer

The LANSA Composer software is available already translated into several languages. In addition it contains features that allow definitions to be created with limited multilingual support. This appendix describes these features under the following headings:

LANSA Composer Product Translations Multilingual support in LANSA Composer

### **LANSA Composer Product Translations**

The core LANSA Composer software is translated into and can be executed in the following languages:

- English
- French
- Japanese

The following section contains information on how to start LANSA Composer in any of the languages for which translations are supplied:

Specifying the language to start LANSA Composer

The LANSA Composer Guide (this guide) is supplied translated into:

- English
- Japanese

The LANSA Composer client software will automatically open the appropriate language version of the LANSA Composer Guide according to the language with which the software was started.

Certain supplied LANSA Composer definitions, such as the set of supplied Activities are available with descriptions and notes translated into the following languages:

- English
- French
- Japanese

The mapping tool supplied and used with LANSA Composer is Altova MapForce. This software is available in the following languages:

- English(supplied on the LANSA Composer product media)
- Japanese(supplied on the LANSA Composer product media)
- German(available for download from Altova's web-site)

### Specifying the language to start LANSA Composer

You can only start LANSA Composer using the languages for which translations are provided, presently:

- English
- French
- Japanese

### Specifying the language to start LANSA Composer client software

When LANSA Composer client software starts, by default it will start in the language appropriate for the computer on which it is running. For example, on a French computer, LANSA Composer will start in French, or on a Japanese computer, LANSA Composer will start in Japanese.

It is possible to override this behaviour to start LANSA Composer in a different language using a command-line switch to dxstart.exe or by modifying the dxstart.cfg configuration file. For more information, refer to:

Using dxstart.exe to start functions of LANSA Composer on Windows

Server functions (such as a Processing Sequence run or the Database Housekeeping task) that are initiated from the LANSA Composer client software will run on the server in the language used to start the LANSA Composer client.

# Specifying the language to run a LANSA Composer Processing Sequence

When you initiate a Processing Sequence run on the server system, you can specify the language in which LANSA Composer will execute. This affects, for example, the language for messages, especially for the Processing Sequence log.

For more information, refer to:

Run a Processing Sequence using the COMPOSER command

Run a Processing Sequence using a Run Control File

## Specifying the language to start LANSA Composer IBM i server software

On IBM i servers, you can use the COMPOSER command to invoke selected LANSA Composer functionality, including to run a Processing Sequence. The COMPOSER command provides a parameter that allows you to specify the

language in which LANSA Composer will execute. For more information, refer to:

The COMPOSER command for IBM i servers

### **Multilingual support in LANSA Composer**

Irrespective of the language in which LANSA Composer windows, menu and messages are shown, you can, of course, use your local language when entering, displaying and printing LANSA Composer definitions such as Activities, Configurations, Transformation maps and Processing Sequences (providing the client and server systems both support the character sets or code-pages required and provide the necessary client-server translations).

But in addition, LANSA Composer provides limited support for holding such definitions in more than one language. If used, this would mean, for example, that a user running LANSA Composer in English could see English descriptions and notes for definitions such as Activities, Configurations, Transformation maps and Processing Sequences, while another user connected to the same LANSA Composer server system could see the same definitions with French descriptions and notes.

This multilingual support is presently provided ONLY for the following parts of LANSA Composer definitions:

- descriptions for each definition
- notes associated with each definition

When you create new definitions in LANSA Composer, the description and notes entered are saved as a "base" language. If the current language is not English they also saved as a "translation" for the current language.

If you wish to enter translations for these items, then you must start the LANSA Composer client again in the alternate language and re-enter the descriptions and notes in that language. Doing so will not alter the descriptions and notes that were originally entered and saved as the "base" language. The descriptions and notes will be retained in the LANSA Composer database for both languages.

If a translation is not present for an item for the current language, LANSA Composer will display the description or notes for the "base" language. The "base" language is implicitly the language with which the definition was initially created.

Note however, that for the purpose of subsequent modification, the "base" language is assumed to be English. That is, you can only modify the "base" language version of the description and notes when running the LANSA Composer client in English. (Remember, if LANSA Composer was running in a language other than English when you created the definitions, then the description and notes were saved as both the "base" language and for the current language. If you subsequently modified the definitions while running LANSA Composer client in the same language, then only the second set were modified in the database - the "base" language description and notes were not modified.)

### Scope and purpose of the Multilingual support

The multilingual support described above is mainly intended to allow LANSA to provide supplied Activity and other definitions with descriptions and notes in the languages for which we also provide LANSA Composer product translations, viz:

- English
- French
- Japanese

While running LANSA Composer in any one language, the multilingual features are not apparent and you will not be aware of the presence (or absence) of translations of the LANSA Composer definitions for other languages. LANSA Composer only displays, prints and uses the translations that apply to the current language.

LANSA expects that most customer implementations would not require the multilingual support and would not enter translations in multiple languages in connection with their own customer-created definitions.

### **Deployment Considerations for Multilingual Definitions**

If you do create translations in multiple languages for your own definitions, there are special considerations for deploying those definitions to other LANSA Composer installations.

LANSA Composer provides Export and Import features to support the deployment of LANSA Composer solutions from one LANSA Composer installation to another. For a fuller description of these features and considerations for their use, refer to:

```
Deploying Solutions for LANSA Composer
```

The standard Export and Import features will deploy the complete definitions of selected items in almost every respect. However, for the descriptions and notes,

the standard deployment is limited to only the "base" language (as described above) and the translation for the current language, if present.

If translations for these items have been entered for other languages, they must be separately deployed using the procedure outlined below. This procedure is intended primarily for use by LANSA to create the deployment materials for definitions such as Activities that are supplied with LANSA Composer.

The procedure necessitates modifying the Windows registry on the computer that performs the Export. Such modifications should be performed with great care by suitable qualified individuals - incorrect Windows registry modifications can have serious consequences including rendering the computer inoperable.

- 1. On the source system:
  - a) Export the required definitions in the normal way

Descriptions and notes for the selected items will be exported for the "base" language and the current language, if present.

b) Edit the Windows registry to enable the *Export Translations* feature

In the key:

HKEY\_CURRENT\_USER\Software\LANSA Composer\<*root*>\ <*server*>\Settings

create a new string value with the name '*AllowXLateExport*' and set its value to '*Y*'.

c) Start the LANSA Composer client on a computer that is capable of supporting the character set used by the language whose translations you wish to export. For example, to export Japanese translations, you will typically have to perform the export on a Japanese language PC.

NOTE: Data corruption and/or program failure arising from encoding errors may arise if the exporting computer is not capable of correctly representing the characters used for the translation language being exported.

d) Select *Export* in the *Navigator* and begin by selecting the items whose

translations you wish to export in the normal way. But then expand the *Translations* item and select the language(s) for which you wish to export translations for the selected items.

e) Click the *Export* button and complete the export in the usual way.

Only the translations of the descriptions and notes for the selected items and languages will be exported. This export file does not contain the complete item definition - it can only be used to deploy the translations when the items already exist on the target system.

- f) You may remove the registry entry above or set it to 'N' to disable the *Export Translations* feature.
- 2. On the target system:
  - a) Load / import the item definitions using the export file created in step 1(a)

The import will completely replace the items on the target system if they already exist. This includes removing any existing translations of descriptions and notes. The import will load the "base" language descriptions and notes; it will also the load the "current" language translations, if the language is the same as the exporting system.

b) Start the LANSA Composer client on a computer that is capable of supporting the character set used by the language whose translations you wish to import. For example, to import Japanese translations, you will typically have to perform the import on a Japanese language PC.

NOTE: Data corruption and/or program failure arising from encoding errors may arise if the importing computer is not capable of correctly representing the characters used for the translation language being imported.

c) Load / import the item translations using the export file created in step 1(e)

The import will load translations only for items that already exist in the target system.