WebSphere Process Server V6

FDL2BPEL Conversion

Version 6.1

Migration of WebSphere MQ Workflow source artifacts to WebSphere Process Server

Third Edition (February 2008)

This book is edited by International Business Machines Corporation, USA
© Copyright International Business Machines Corporation 2004, 2008

The text is subject to alteration.

Published by:

IBM Deutschland Entwicklung GmbH
Department 1892
Schoenaicher Strasse 220
71032 Boeblingen

Related Publications

IBM WebSphere Process Server

- Version 6.1 Information Center at http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp
- Kurt Lind, "Authorization and staff resolution in Business Process Choreographer: Part 1: Understanding the concepts and components of staff resolution" (available at http://www.ibm.com/developerworks/websphere/techjournal/0710_lind/0710_lind.html)
- Kurt Lind, "Authorization and staff resolution in Business Process Choreographer: Part 2: Understanding the programming model for staff resolution" (available at http://www.ibm.com/developerworks/websphere/techjournal/0711_lind/0711_lind.html)
- More links to related WPS publications at: http://www.ibm.com/developerworks/websphere/zones/was/wpc.html

IBM WebSphere MQ Workflow

- IBM WebSphere MQ Workflow: Concepts and Architecture, Version 3.6, SH12-6285
- IBM WebSphere MQ Workflow: Getting Started with Buildtime, Version 3.6, SH12-6286
- IBM WebSphere MQ Workflow: Programming Guide, Version 3.6, SH12-6291
- IBM WebSphere MQ Workflow: Administration Guide, Version 3.6, SH12-6289

BPEL

- Business Process Execution Language for Web Services Version 1.1 (available at http://www.ibm.com/developerworks/library/specification/ws-bpel/)
- Web Services Business Process Execution Language Version 2.0 (available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html)

CONTENTS

Figures

Tables

Abbreviations

API	Application Programming Interface
BFM	Business Flow Manager
BPC	Business Process Choreographer
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
FDL	Flow Definition Language
HTM	Human Task Manager
ID	Identifier
JAR	Java Archive
JDK	Java Development Kit
JMS	Java Messaging Service
LDAP	Lightweight Directory Access Protocol
MQ	Message Queuing
MQMD	MQ Message Descriptor
PEA	Program Execution Agent
PES	Program Execution Server
RFH2	Rules and Formatting Header 2
SCA	Service Component Architecture
SCDL	Service Component Definition Language
SOA	Service-Oriented Architecture
TEL	Task Execution Language
UPES	User-defined Process Execution Server
WID	WebSphere Integration Developer
WMQWF	WebSphere MQ Workflow
WPS	WebSphere Process Server
WSDL	Web Service Definition Language
XML	Extended Markup Language
XPath	XML Path Language
XSD	XML Schema Definition

What is new with Version 6.1?

Initial WID process view without fully expanded structured activities

Unlike in WMQWF Buildtime, the business process editor in the former versions of WID expanded every BPEL structured activity[1] by default. As a consequence, you could not compare the topological structure[2] of the original Flow Definition Language (FDL) process with the structure of the corresponding BPEL process. Now, the initial view of the business process editor shows the BPEL process with collapsed structured activities. This feature offers an improved usability of the business process editor:

1. When opening a business process model for the first time, only the first level of a hierarchical organized process model is expanded by default. All contained structured activities are shown as collapsed node icons.

2. Similar to WMQWF Buildtime, you can discover the hierarchical structure of your process by stepwise expanding the collapsed nodes. Each step will expand just the next layer of the model hierarchy.

3. The last view that you see being displayed in the graphical model editor will be preserved, if you save the process in the editor.

Support of *data type variables*

Former versions of the FDL2BPEL Conversion Tool mapped WMQWF data containers to BPEL variables that were marked as *interface variables*. This migration policy established challenges when you wanted to rework the migrated model by optimizing the model structure and by eliminating BPEL variables. Now, FDL2BPEL generates *data type variables* by default. As a consequence, the BPEL variables lose their strong coupling to the "interfaces" and can be removed or reassigned to another BPEL activity, if needed.

Support document/literal wrapped WSDL interfaces

This version of the FDL2BPEL Conversion Tool introduces the document/literal wrapped interfaces to the FDL to WSDL mapping rules. These are the technical characteristics of the document/literal wrapped interface:

- The input message has a single part.
- The part is an element.
- The element has the same name as the operation.
- The element's complex type has no attributes.

Among other advantages the document/literal wrapped pattern makes the generated WSDL files WS-I compliant. For more details see the article "Which style of WSDL should I use" by Russel Butek at http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/.

Support for shorter variable names

All BPEL variable names generated by the FDL2BPEL Conversion Tool consist of the concatenation of the name of the process or of an activity and a name suffix that indicates the role of an input or output container.[3] Since FDL activity names only need to be unique within

[1] BPEL activities such as "Scope", "Parallel Activities", "Sequence", and "While Loop".

[2] See "*topology preservation policy*" in section "Are the migrated models too complex?" on page 16.

[3] See "Mapping FDL data container to BPEL" on page 54.

the scope of a process (or block), previous versions have used a name prefix that helped to avoid name clashes by using qualified names. The long name prefix consisted of a sequence of FDL process and activity names that reflected the position of the migrated data container in the model hierarchy. The FDL2BPEL Conversion Tool now resolves any name clashes immediately and generates short but unique names.[4]

Support for activity expiration

Source artifact migration from WMQWF to WPS was limited with respect to using activity state expressions in exit or transition conditions.[5] The documentation of FDL2BPEL Conversion Tool V6.0.2 explained how to overcome the limitation by adding fault handlers to the respective BPEL activities. The current version adds such fault handlers automatically.[6]

Option to not create predefined data members as part of variables

The command line version of the previous FDL2BPEL Conversion Tool already had a parameter **–pn** ("Do not create predefined data members")[7] that offered you the option to exclude predefined members of WMQWF data containers from being migrated to BPEL. Now this option is also available as a corresponding check box "Create predefined data members" that you can deselect, if you want to get rid of predefined data members in BPEL variables.

Staff assignment policies

The FDL2BPEL Conversion Tool V6.1 supports the new human task feature participant support. It migrates the following staff assignment policies that are newly introduced in V6.1[8]:

- Prefer not absent users
- Assign substitute if user is absent

Mapping of the WMQWF property *Autonomy*

The FDL2BPEL Conversion Tool maps value "control" of the FDL process *autonomy* flag to the corresponding BPEL autonomy value "peer". In all other cases, the BPEL autonomy flag is set to "child".[9]

Mapping of the WMQWF property *Execution mode*

In case of a UPES activity the FDL property *execution mode* (with the selectable values "synchronous" or "asynchronous") is mapped to the corresponding SCA Import property "Preferred interaction style".[10]

Improved error reporting

Though the current version of FDL2BPEL Conversion is still expecting a syntactically and semantically correct FDL input file, the tool now supports a limited validation for completeness. That is, the tool scans the FDL for indispensable definitions of subprocesses, data structures, programs, and UPES servers. If the tool encounters severe errors, it reports them with a message and does not start the migration operation itself. If there are no errors, the

[4] See "Mapping of FDL names to BPEL" on page 35.

[5] See "Activity states" on page 133.

[6] See "Activity expiration" on page 75.

[7] Note that the options **-pi** and **-pn** cannot be used together (see "Running the FDL2BPEL Conversion Tool" on page 27).

[8] See "Substitution" on page 90.

[9] See "Autonomy" on page 108.

[10] See "Preferred interaction style" on page 96.

WID Migration Wizard will create a module. Most errors and warnings locate the problem in the FDL by a line number.

Continue On Error setting

The "Continue on Error" setting determines how the process should proceed when a fault (on either an Invoke or Snippet activity) is not caught on the enclosing scope, or handled through a local fault handler. The FDL2BPEL Conversion Tool disables the "Continue on Error" setting. Accordingly, each failing activity without a fault handler is put into the stopped state, and a work item for the process administrator(s) is created so that the problem can be repaired. This corresponds to the behavior of WMQWF business processes.

Chapter 1: Introduction

Subject of this document

This document describes how to use the ***FDL2BPEL Conversion Tool Version 6.1***. The tool converts FDL definitions of business process models exported from the Buildtime component of IBM® WebSphere® MQ Workflow Version 3.6.0 (WMQWF) into corresponding BPEL definitions of business process models which you can import into IBM WebSphere Integration Developer (WID). It generates XML artifacts that you need to deploy and execute these processes with Business Process Choreographer (BPC) of IBM WebSphere Process Server Version 6.1 (WPS). The generated XML definition files include XML schema definitions for data flow objects, WSDL, BPEL, TEL, and SCDL definitions of SCA components and imports.

Who are the readers that will most benefit from this document?

This documentation is limited to the technical concepts of the FDL2BPEL Conversion Tool that you need to be familiar with in order to understand its working, how to use it, and how to benefit from its capabilities. You should have expert level knowledge about WebSphere MQ Workflow concepts and at least solid knowledge about Web services concepts and Business Process Choreographer. This document does not provide advice for planning the migration from an in-production WMQWF system to a WPS system. However, reading this document and getting somewhat familiar with the FDL2BPEL Conversion Tool is a good preparation before starting the planning phase.

Migrating with or without a tool?

A fundamental decision, that you must make, is to choose between a semi-automatic migration supported by this tool and a manual migration by developing new BPEL business process models "from scratch". The term "semi-automatic" means that you cannot expect the migrated business process models to deploy and execute without some manual reworking. There are some possible reasons for this:

- The generated artifacts might be suboptimal because you detect redundant process network nodes and BPEL variables that you should eliminate to achieve better performance.
- The generated BPEL business process(es) will be incomplete if not all FDL attributes could be translated due to limitations of the tool or insurmountable differences of model concepts and programming model.

Nevertheless, you can expect that using the FDL2BPEL Conversion Tool and manually reworking the migrated business process models will be less expensive and less error-prone than doing the migration without the tool. You will experience that it is a complex task to migrate the overall model structure, but getting the manual translation of data and control flow right can be cumbersome and boring!

Are the migrated models too complex?

The generated BPEL business process models look more complex than the model graphs that you are familiar with when using WMQWF Buildtime. Here are some simple reasons for this:

- BPEL does not know data connectors. The FDL2BPEL Conversion Tool translates FDL data connectors into "Snippet" activities.[11]
- BPEL does not know the concept of an activity data input/output container with default value settings. The FDL2BPEL Conversion Tool translates each data input/output

[11] See "Mapping FDL data flow to BPEL" on page 77.

container into corresponding BPEL variables and the setting of default values into an "Assign" activity with name "Set the data default values".[12]

- Some implicit data and control flow semantics of FDL must be modeled explicitly with BPEL. For instance, the business processes migrated to BPEL will contain extra structured activities, such as "Sequence", "Parallel Activities" (aka "Flow"), and "While Loop" activity nodes that are required to ensure the proper flow of data and control.[13] [14]

In conclusion, your impression of increased complexity seems to be confirmed. However, the mapping rules of the FDL2BPEL Conversion Tool follow a fundamental ***topology preservation policy***. That is, despite the addition of extra activities, the structure of the migrated process models is not changed. This helps you when validating the correctness of process model migration and in gaining confidence in the new model's appearance.

Example:

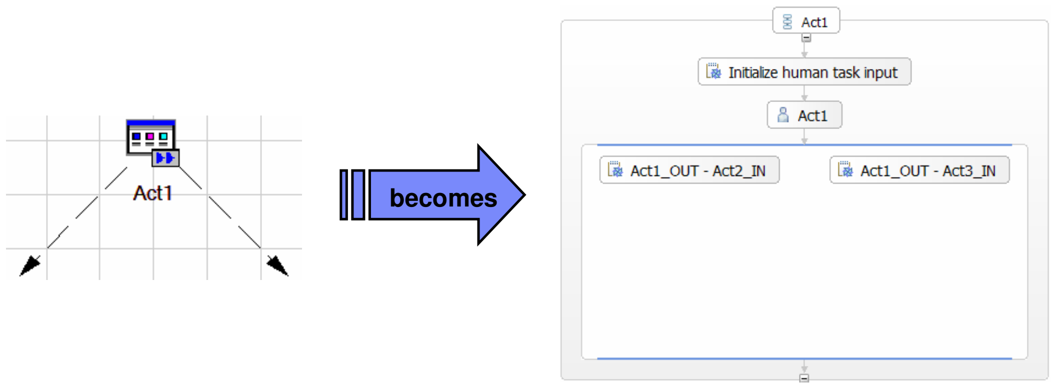


Figure 1: Topology preservation policy (1)

As you see, the FDL activity "Act1" becomes a BPEL activity "Act1", which is embedded in another "Sequence" activity (also named "Act1"). The "Sequence" activity serves as a kind of wrapper for the translation of the activity itself (here: "Human Task"). The two snippets represent the original outbound data connectors of the FDL activity "Act1". To verify that the original topology of activity nodes and control connectors is preserved in the BPEL process model, collapse all these wrappers (BPEL "Sequence" activities) by clicking on the "-" symbol below the "Sequence" icon Act1. Now you should see that the migrated business process contains the same number and sequence of activities:

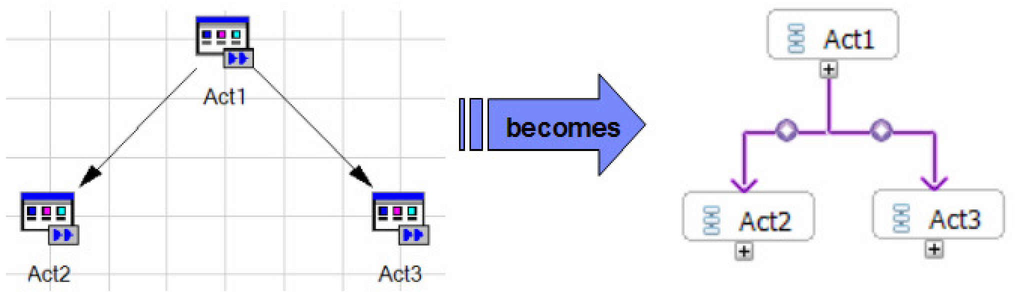


Figure 2: Topology preservation policy (2)

[12] See "Mapping FDL data container to BPEL" on page 54.

[13] See "Mapping the FDL workflow to BPEL" on page 57.

[14] See "Exit conditions" on page 73.

FDL2BPEL usage modes

The FDL2BPEL Conversion Tool Version 6.1 is available in two modes:

1. Command line tool (see page 25).

2. WID Migration Wizard (see page 21).

Both modes of the FDL2BPEL Conversion Tool require a semantically complete FDL definition of a process model that you export from WMQWF Buildtime with the option "Export deep". Using the "Export deep" option ensures that all necessary data, program, and subprocess specifications are included. Make sure that any user-defined process execution server (UPES) definitions that are referenced in your WMQWF process model are also selected when you export the FDL file from WMQWF Buildtime.

What the FDL2BPEL Conversion Tool cannot do for you

The FDL2BPEL Conversion Tool does not cover the migration of the following:

- WMQWF runtime instances
- WMQWF program applications that are invoked by a WMQWF P*rogram Execution Agent* (PEA) or WMQWF *Process Execution Server* (PES for z/OS®)
- WMQWF network hierarchy
- WMQWF staff
- Program applications that use a WMQWF API
- WMQWF auditing

In addition, the tool may not work properly with FDL input files that have a version ID prior to V3R6 (see the limitation "Migration of early FDL versions" on page 130).

Some best practice hints

The scope and completeness of the mapping depends on how far you adhere to the following "best practices" guidelines for migration:

- Make sure that all FDL program activities are associated with a UPES, if they are not pure "staff" activities.
- Make sure that all staff assignments for WMQWF program activities are compliant with the BPC Task Execution Language (TEL) default "people assignment criteria".[15]
- Prefer short and simple names in order to improve the readability of migrated process models. Note that some valid FDL names may be illegal BPEL names. The FDL2BPEL Conversion Tool automatically converts problematic FDL names to valid BPEL names.

The FDL2BPEL Conversion Tool produces syntactically correct BPEL constructs, even for non-migratable FDL constructs (PEA or PES program activities, some dynamic staff assignments etc.), which need manual adaption to executable BPEL artifacts.

Mapping rules (overview)

Table 1 outlines the mapping rules:

[15] See "Mapping WMQWF staff assignment criteria to TEL people assignment criteria" on page 85

Table 1: Mapping rules (overview)

	WMQWF Process Model Construct	BPEL with Extensions Construct[16]
Structure	Process	*Process with execution mode*: *longRunning* (BPEL extension) *Partner links* for inbound and outbound interfaces of process
	Program activity	*Invoke* activity
	Process activity	*Invoke* activity
	Empty activity	*Empty* activity
	Block	*Parallel activities (also called flow* activity)
Control Flow	Start condition of activity	*Join condition* of activity
	Exit condition of activity	*While* activity (enclosing the actual activity)
	Control connector	*Link*, *source* element of activity, *target* element of activity
	Transition condition (of control connec-tor)	*Transition condition* (assigned to *source* element)
	Staff assignment of activity	*Human task* activity (BPEL extension using assigned TEL task definition)
	Notification	*Escalation* (BPEL extension using assigned TEL task definition)
	Expiration for activity	*Expiration* for invoke activity (BPEL extension)
Data Flow	Source and sink	*Variables* for process input and process output *Receive* activity and *reply* activity
	Input / output container of activity	*Variables* for specification of input/output of *invoke* activity
	Data connector	*Snippet (*activity with inline Java code that copies data from a source BPEL variable to a target BPEL variable)
	Default data settings	*Assign* activity (assigns default data as literal values to a BPEL variable)
	Global data container	*Variable*

After the WID import you should review and, if necessary, modify the generated files. In case of errors, warning or information messages, you will find comment annotations in the generated WSDL and BPEL files. Additional effort may be necessary to either make a successful migration possible or to complete the migration task.

Ideally, make your first migration experiences with small projects. The FDL2BPEL Conversion Tool will simplify the conversion of your FDL process models into BPEL process models, but you should be aware that FDL and BPEL cannot be mapped one-to-one because of differences between the programming models. The semantic scopes of FDL and BPEL share an area of intersection, but they do not overlap completely.

[16] Note that FDL2BPEL translates to a BPEL specification that is based on the BPEL4WS V1.1 standard (cp. http://www.ibm.com/developerworks/library/specification/ws-bpel/) and IBM extensions (for instance Java Snippets).

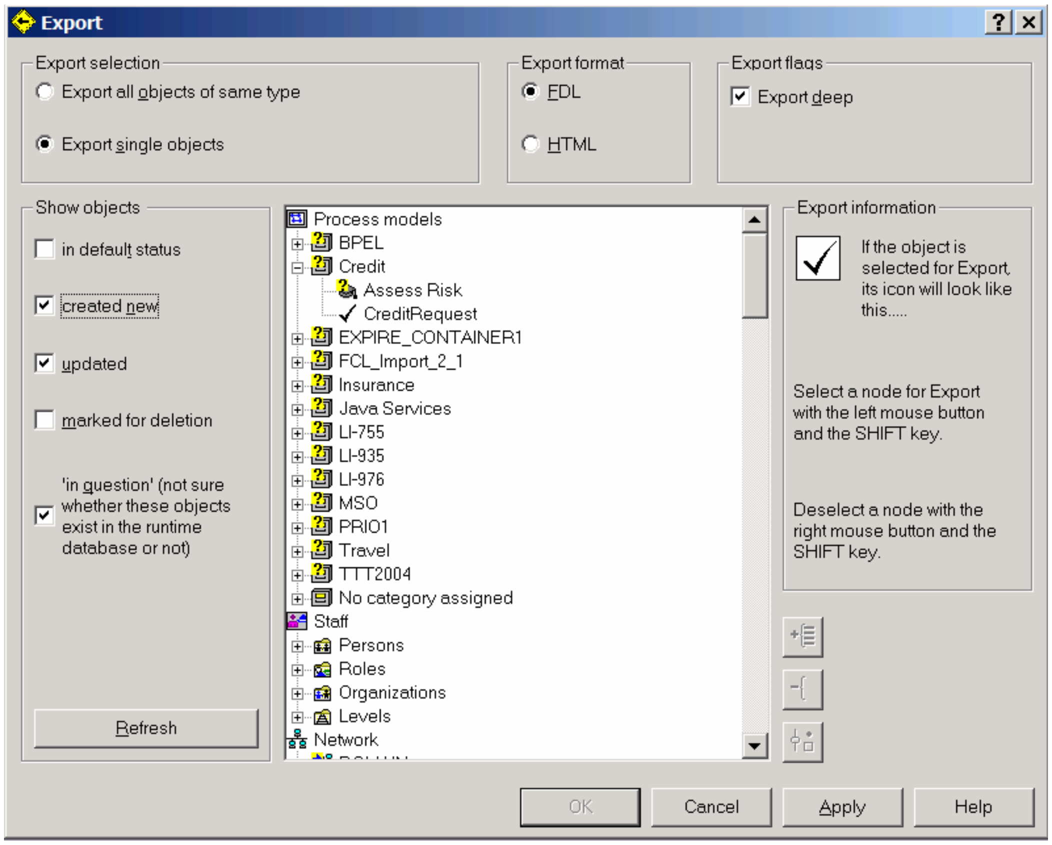
Chapter 2: Using the FDL2BPEL Conversion Tool

Exporting a WMQWF model from WMQWF Buildtime

To create an FDL file, perform the following:

1. Export the WMQWF process model from WMQWF Buildtime with the option "Export deep" (see Figure 3). This ensures that the export includes all referenced objects. For instance, the exported FDL file will also contain the definitions of subprocesses, if your selected process model contains "process activities". This way, you only need to select the top-level process model and the referenced UPES definitions (see below) to export.
2. Your FDL must also contain the definitions of any user-defined program execution server (UPES) that is associated with any program activity in your FDL process. **You must select a UPES definition explicitly for export from WMQWF Buildtime (The option "Export deep" is not sufficient for this purpose.).**

Note that any FDL processing actions, such as **REPLACE**, **UPDATE**, and **DELETE** will be ignored.[17] [18] Also note that the FDL2BPEL Conversion Tool expects a syntactically correct and complete FDL input file.[19] If the tool encounters severe errors, it will report them without starting the migration operation.



Export

Export selection: Export all objects of same type [✓ Export single objects]

Export format: [✓ FDL] HTML

Export flags: [✓ Export deep]

Show objects:
- ☐ in default status
- ☑ created new
- ☑ updated
- ☐ marked for deletion
- ☑ 'in question' (not sure whether these objects exist in the runtime database or not)

Refresh

Process models
- BPEL
- Credit
  - Assess Risk
  - ✓ CreditRequest
- EXPIRE_CONTAINER1
- FCL_Import_2_1
- Insurance
- Java Services
- LI-755
- LI-935
- LI-976
- MSO
- PRIO1
- Travel
- TTT2004
- No category assigned

Staff
- Persons
- Roles
- Organizations
- Levels

Network

Export information

If the object is selected for Export, its icon will look like this.....

Select a node for Export with the left mouse button and the SHIFT key.

Deselect a node with the right mouse button and the SHIFT key.

Figure 3: Exporting a WMQWF model from WMQWF Buildtime

[17] FDL files exported from WBI Modeler should be imported into WMQWF Buildtime before.

[18] See "FDL processing actions" on page 130.

[19] See "Problem detection" on page 131.

Using as WID Migration Wizard

By importing a WMQWF business process model exported from WMQWF Buildtime[20] the WID Migration Wizard converts the FDL definitions into corresponding WPS artifacts. The generated files comprise XML schema definitions for business objects, WSDL definitions, BPEL, SCA import and component definitions, and TEL definitions (see "Understanding the generated files" on page 32). The files are automatically moved into a newly generated Business Integration module.

To use the WID Migration wizard, perform the following:

1. Prepare the WMQWF business process(es) you want to migrate by creating an FDL file (cp. "Exporting a WMQWF model from WMQWF Buildtime" on page 20).

2. Start the WID.

3. In WID, invoke the wizard by selecting **File** > **Import…** > **Business Integration** > **WebSphere MQ Workflow FDL File**

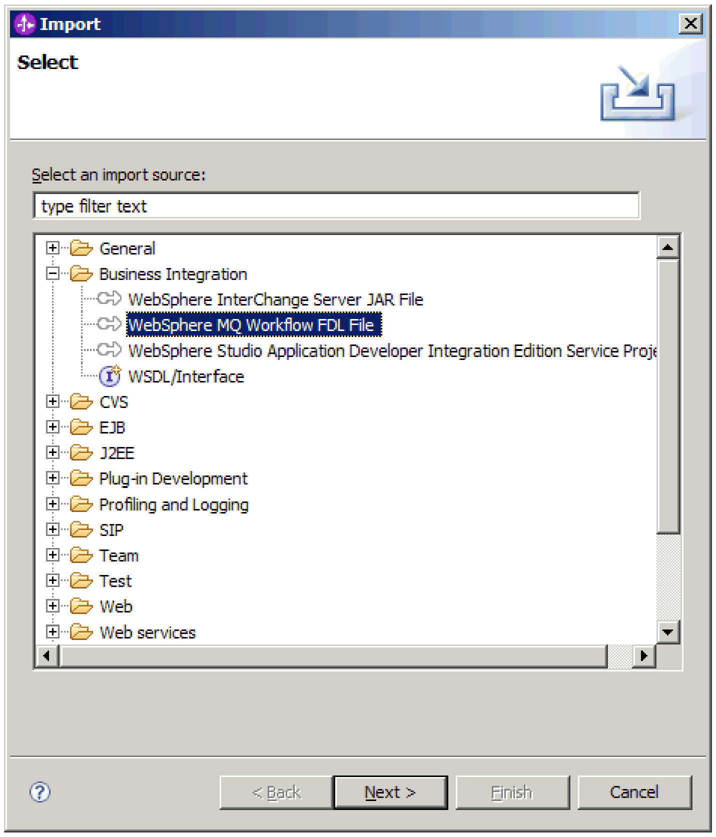
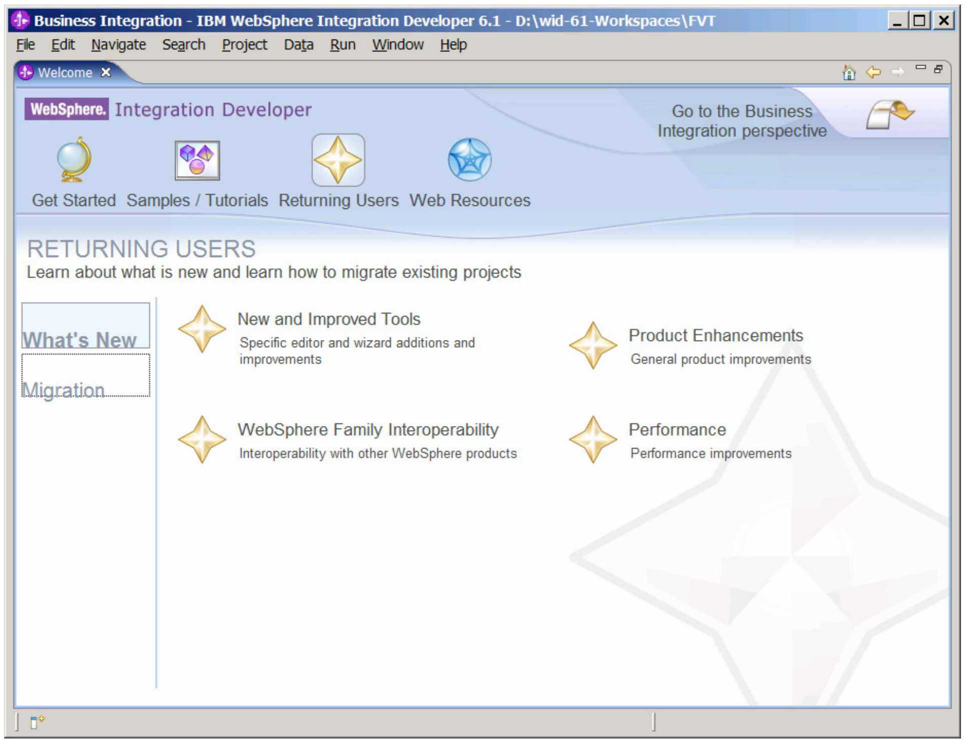


Figure 4: Running FDL2BPEL as WID Migration Wizard (1)

You can also open the WID Migration Wizard from the WID Welcome page by clicking on the **Returning Users** icon to open the Returning Users page. Note that you can return to the Welcome page by clicking on **Help** > **Welcome**.

[20] See "Exporting a WMQWF model from WMQWF Buildtime" on page 20.


Business Integration - IBM WebSphere Integration Developer 6.1 - D:\wid-61-Workspaces\FVT
File Edit Navigate Search Project Data Run Window Help
Welcome
WebSphere. Integration Developer
Go to the Business Integration perspective
Get Started
Samples / Tutorials
Returning Users
Web Resources
What's New
Migration

RETURNING USERS
Learn about what is new and learn how to migrate existing projects

New and Improved Tools
Specific editor and wizard additions and improvements

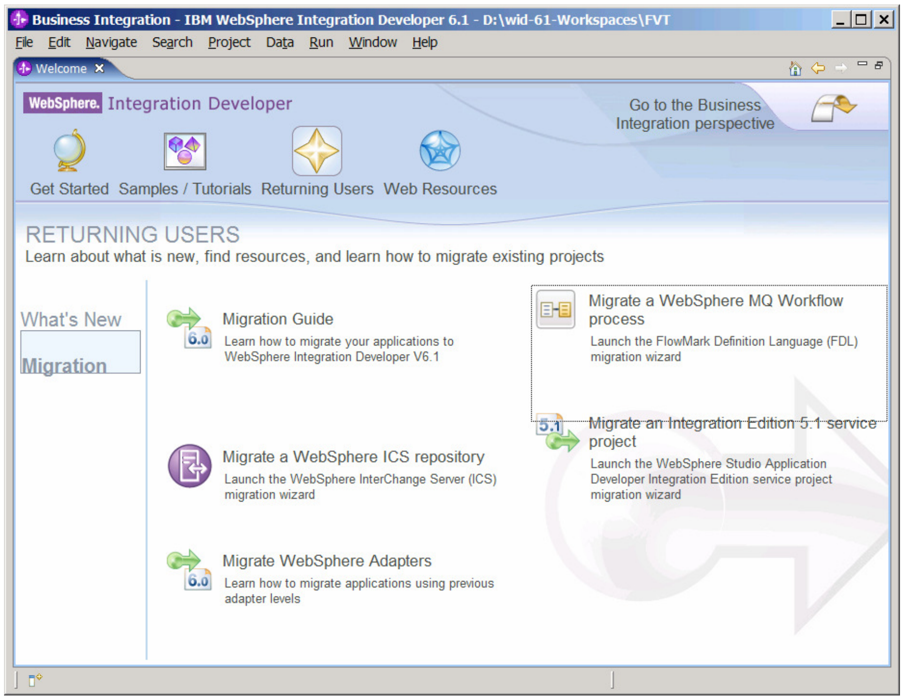
Product Enhancements
General product improvements

WebSphere Family Interoperability
Interoperability with other WebSphere products

Performance
Performance improvements

Figure 5: Running FDL2BPEL as WID Migration Wizard (2)

Click **Migration** on the left of the Returning Users page to open the Migration page:


Business Integration - IBM WebSphere Integration Developer 6.1 - D:\wid-61-Workspaces\FVT
File Edit Navigate Search Project Data Run Window Help
Welcome
WebSphere. Integration Developer
Go to the Business Integration perspective
Get Started
Samples / Tutorials
Returning Users
Web Resources
What's New
Migration

RETURNING USERS
Learn about what is new, find resources, and learn how to migrate existing projects

Migration Guide
Learn how to migrate your applications to WebSphere Integration Developer V6.1

Migrate a WebSphere MQ Workflow process
Launch the FlowMark Definition Language (FDL) migration wizard

Migrate an Integration Edition 5.1 service project
Launch the WebSphere Studio Application Developer Integration Edition service project migration wizard

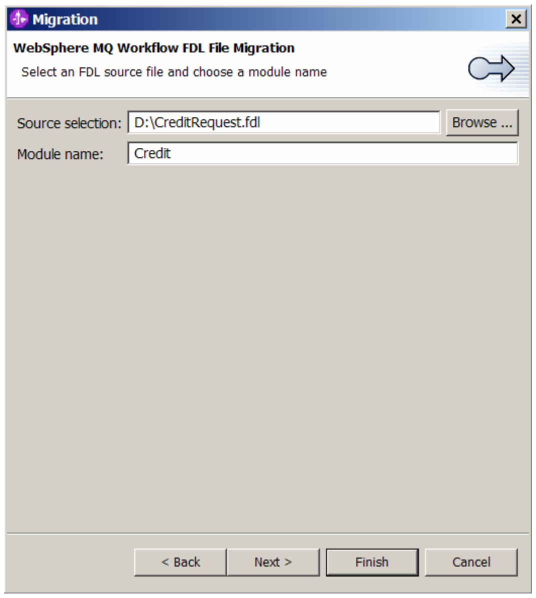
Migrate a WebSphere ICS repository
Launch the WebSphere InterChange Server (ICS) migration wizard

Migrate WebSphere Adapters
Learn how to migrate applications using previous adapter levels

Figure 6: Running FDL2BPEL as WID Migration Wizard (3)

From the Migration page, select the option "Migrate a WebSphere MQ Workflow process".

4. The WID Migration Wizard opens. Either enter the absolute path and name of the FDL file into the Source selection field or find the FDL file by clicking **Browse…** and navigating to the file.[21] Enter the module name in the Module name field, and then click **Next** >.



WebSphere MQ Workflow FDL File Migration

Select an FDL source file and choose a module name

Source selection: D:\CreditRequest.fdl

Module name: Credit

Figure 7: Running FDL2BPEL as WID Migration Wizard (4)

5. The "Migration Options" page opens. Here you can accept the migration defaults or change the options. If you prefer to have the name conversion from FDL to BPEL under your own control, select the "Treat name conflicts as errors" check box.[22] Otherwise, FDL2BPEL will solve any name conflicts automatically. If you have no need for using predefined data members, deselect option "Create predefined data members".[23] The "Initialize predefined data members" check box adds extra nodes to the process to initialize predefined data members[24]. Moreover, you can overwrite the target namespaces for XM Schema[25], WSDL[26], and BPEL[27] definitions.

[21] This step corresponds to entering the **-i** argument if you start the FDL2BPEL Conversion Tool from the command line.

[22] This check box corresponds to the **-fc** option if you start the FDL2BPEL Conversion Tool from the command line.

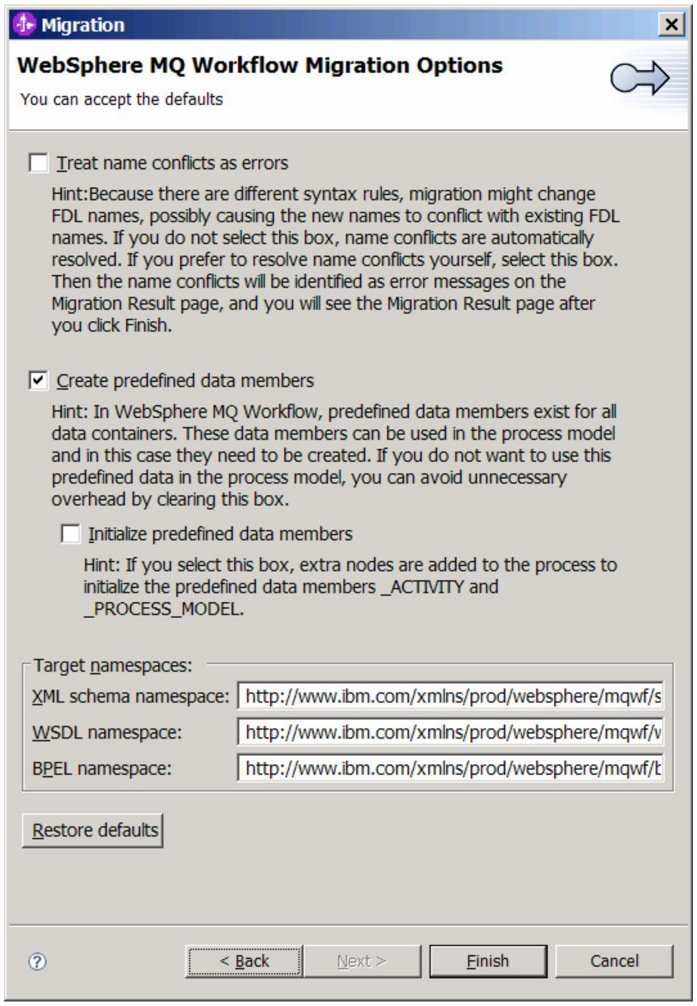
[23] This check box corresponds to the **-pn** option if you start the FDL2BPEL Conversion Tool from the command line.

[24] This check box corresponds to the **-pi** option if you start the FDL2BPEL Conversion Tool from the command line.

[25] This entry field corresponds to the **-tx** argument if you start the FDL2BPEL Conversion Tool from the command line.

[26] This entry field corresponds to the **-tw** argument if you start the FDL2BPEL Conversion Tool from the command line.

[27] This entry field corresponds to the **-tb** argument if you start the FDL2BPEL Conversion Tool from the command line.



Migration

WebSphere MQ Workflow Migration Options

You can accept the defaults

[ ] Treat name conflicts as errors

Hint:Because there are different syntax rules, migration might change FDL names, possibly causing the new names to conflict with existing FDL names. If you do not select this box, name conflicts are automatically resolved. If you prefer to resolve name conflicts yourself, select this box. Then the name conflicts will be identified as error messages on the Migration Result page, and you will see the Migration Result page after you click Finish.

[✓] Create predefined data members

Hint: In WebSphere MQ Workflow, predefined data members exist for all data containers. These data members can be used in the process model and in this case they need to be created. If you do not want to use this predefined data in the process model, you can avoid unnecessary overhead by clearing this box.

[ ] Initialize predefined data members

Hint: If you select this box, extra nodes are added to the process to initialize the predefined data members _ACTIVITY and _PROCESS_MODEL.

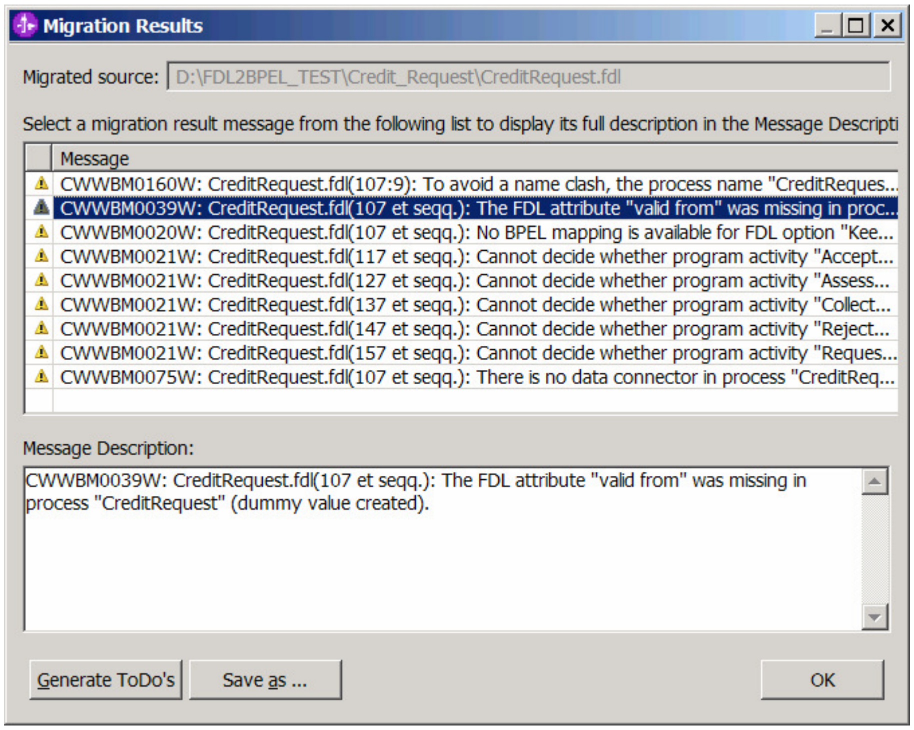
Target namespaces:

XML schema namespace:	http://www.ibm.com/xmlns/prod/websphere/mqwf/s
WSDL namespace:	http://www.ibm.com/xmlns/prod/websphere/mqwf/v
BPEL namespace:	http://www.ibm.com/xmlns/prod/websphere/mqwf/t

Figure 8: Running FDL2BPEL as WID Migration Wizard (5)

Click **Finish**.

1. If the migration produces no errors, warnings, or informational messages, the wizard window will close. Otherwise, a **Migration Results** window appears containing the messages, as shown below:



Migrated source: D:\FDL2BPEL_TEST\Credit_Request\CreditRequest.fdl

Select a migration result message from the following list to display its full description in the Message Descripti

Message
CWWBM0160W: CreditRequest.fdl(107:9): To avoid a name clash, the process name "CreditReques...
CWWBM0039W: CreditRequest.fdl(107 et seqq.): The FDL attribute "valid from" was missing in proc...
CWWBM0020W: CreditRequest.fdl(107 et seqq.): No BPEL mapping is available for FDL option "Kee...
CWWBM0021W: CreditRequest.fdl(117 et seqq.): Cannot decide whether program activity "Accept...
CWWBM0021W: CreditRequest.fdl(127 et seqq.): Cannot decide whether program activity "Assess...
CWWBM0021W: CreditRequest.fdl(137 et seqq.): Cannot decide whether program activity "Collect...
CWWBM0021W: CreditRequest.fdl(147 et seqq.): Cannot decide whether program activity "Reject...
CWWBM0021W: CreditRequest.fdl(157 et seqq.): Cannot decide whether program activity "Reques...
CWWBM0075W: CreditRequest.fdl(107 et seqq.): There is no data connector in process "CreditReq...

Message Description:

CWWBM0039W: CreditRequest.fdl(107 et seqq.): The FDL attribute "valid from" was missing in process "CreditRequest" (dummy value created).

Figure 9: Running FDL2BPEL as WID Migration Wizard (6)

In the **Migration Results** window, you can see the migration messages that were generated during the migration process. By selecting a message from the upper Message list, you can display the complete information about that message in the lower "Message Description" window. To keep all messages for future reference, click the **Generate ToDo's** button to create a list of "ToDo" tasks in the task view, and/or click the **Save as...** button to save the messages as a text file in the file system. Finally, click **OK**.

2. If not already done, change to the WID Business Integration perspective. After having finished the above steps without error messages listed in the Migration Results window[28] you will find a new module in the Business Integration tree view with the name that you entered in step 4:

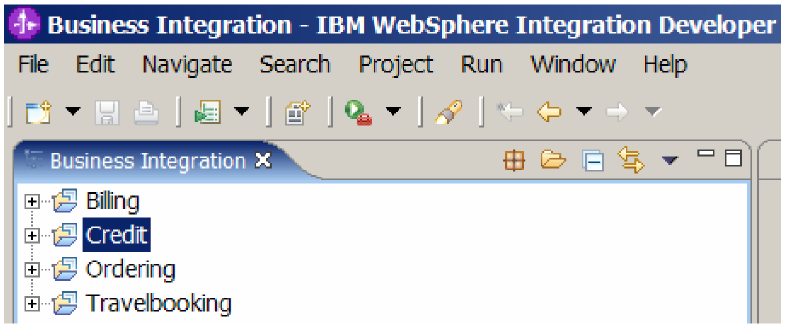


Figure 10: Running FDL2BPEL as WID Migration Wizard (7)

Using as command line tool

As an example, this chapter explains how you can prepare a command file for Microsoft Windows® that allows you to invoke the FDL2BPEL Conversion Tool from the command line.

[28] Warnings are allowed.

Compared with the WID Migration Wizard, the command line version has, despite its minor usability, the advantage that it delivers detailed information about the migration progress. For example, other than the WID Migration Wizard, it reports the renaming of FDL objects (see "Mapping of FDL names to BPEL" on page 35). Maybe you prefer to have more control on the renaming of FDL names that are incompatible with BPEL and decide to perform the renaming in WMQWF Buildtime or in the exported FDL file. This way, you can run the migration several times until you have the expected result and run it before you import the generated files[29] into WID.

Installation

Prerequisites

- IBM WebSphere Integration Developer 6.1 (in order to rework or improve the migrated processes)
- IBM WebSphere Process Server, version 6.1 (in order to deploy and execute the migrated processes)

Preparing the "fdl2bpel" command file

Running the FDL2BPEL Conversion Tool from the command line requires a classpath setting that includes the following JAR files:

- fdl2bpelcore_6.1.0
- CommonMigrationInterface_6.1.0
- core_6.1.0
- common_6.1.0

For this, create a new command file with name "fdl2bpel.bat". Edit the command file, and insert the following code:

```
@echo off
setlocal
@set WID_DIR=<WID installation directory>
@set WID_SHARED_DIR=<WID shared directory>
@set WSTOOLSPLUGINS=%WID_SHARED_DIR%\plugins
@set JDK_HOME=%WID_DIR%\jdk\jre\bin
@set FDL2BPELCORE=%WSTOOLSPLUGINS%\com.ibm.bpe.fdl2bpelcore_<version nbr>.jar
@set MIGRATIONINTERFACE=%WSTOOLSPLUGINS%\com.ibm.wbiserver.migration.CommonMigrationInterface_<version nbr>.jar
@set BPECORE=%WSTOOLSPLUGINS%\com.ibm.bpc.core_<version nbr>.jar
@set BPECOMMON=%WSTOOLSPLUGINS%\com.ibm.bpc.common_<version nbr>.jar
@set FDL2BPELCLASSPATH=%FDL2BPELCORE%;%MIGRATIONINTERFACE%;%BPECORE%;%BPECOMMON%
@echo on
"%JDK_HOME%\java" -classpath "%FDL2BPELCLASSPATH%" com.ibm.bpe.fdl2bpel.converter.FDL2BPELConverter %*
@echo off
@endlocal
```

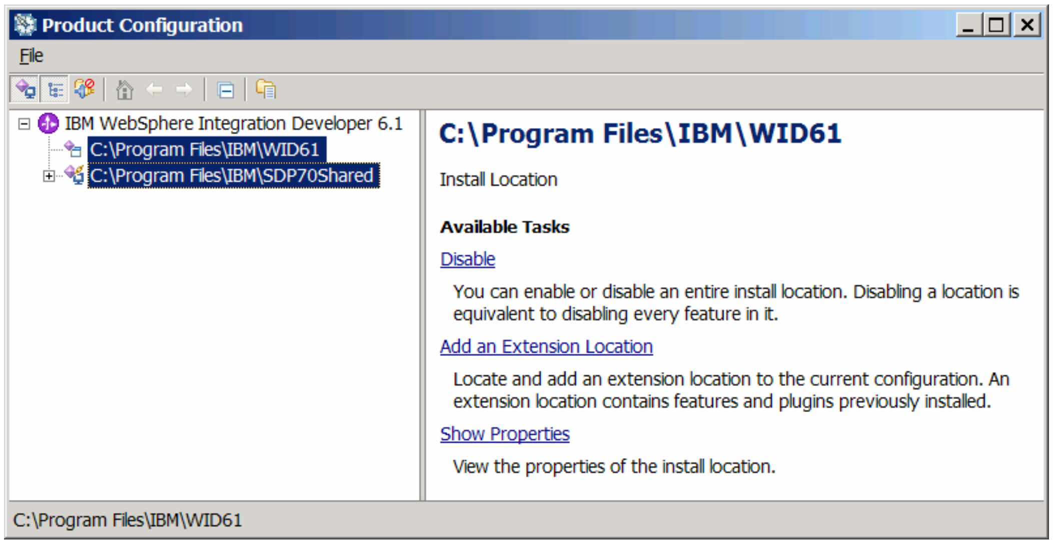
Change the "WID_DIR" and "WID_SHARED_DIR" variable according to your local environment. Examples:

- WID_DIR=C:\Program Files\IBM\WID61
- WID_SHARED_DIR=C:\Program Files\IBM\SDP70SHARED

You can find these directories in you WID product configuration:

[29] See "Chapter 3: Understanding the generated files" on page 32.

1. Click on **Help -> Software Updates -> Manage Configuration**.
2. Look up the two directories in the tree view of the left-hand side (see Figure 11).


Product Configuration
File
IBM WebSphere Integration Developer 6.1
C:\Program Files\IBM\WID61
C:\Program Files\IBM\SDP70Shared

C:\Program Files\IBM\WID61

Install Location

Available Tasks

Disable

You can enable or disable an entire install location. Disabling a location is equivalent to disabling every feature in it.

Add an Extension Location

Locate and add an extension location to the current configuration. An extension location contains features and plugins previously installed.

Show Properties

View the properties of the install location.

C:\Program Files\IBM\WID61

Figure 11: Looking up WID installation directories

At last, search for the referenced Jar files in the plugins directory (referenced by variable WSTOOLSPLUGINS) and change the version numbers accordingly. For example, "com.ibm.bpe.fdl2bpelcore_<version nbr>.jar" becomes "com.ibm.bpe.fdl2bpelcore_6.1.0.200711081700.jar".

How to translate WMQWF models into BPC models

To use the FDL2BPEL Conversion Tool from the command line, first prepare the WMQWF business process(es) you want to migrate by creating an FDL file (see "Exporting a WMQWF model from WMQWF Buildtime" on page 20).

Running the FDL2BPEL Conversion Tool

The FDL2BPEL Conversion Tool has the following command line options:

Table 2: FDL2BPEL command line options

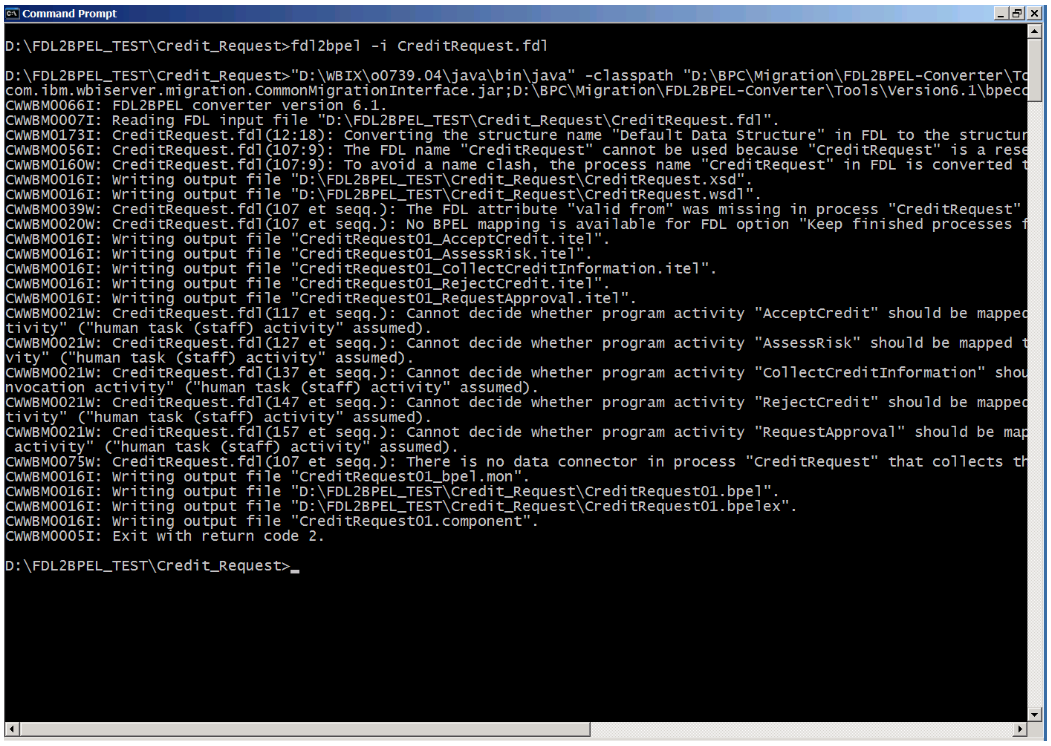
Command line option	Command line argument	Explanation
-h or ?		Displays the help information.
-i	<file name>	Name of the FDL input file.
-od	<directory name>	Location of the directory that all files generated by the FDL2BPEL Conversion Tool will be written to. **Default:** Directory location of the input file
-oe[30]	<project name>	Name of the WID module which you want to import the generated files into to review and generate deploy code. This parameter is required if your FDL input file contains the definition of a process with sub-process invocations (process activities).
-fc	<Boolean>	Treat name conflicts as errors. Because FDL and BPEL have different syntax rules, the FDL2BPEL Conversion Tool might change FDL names to match the BPEL syntax rules. To avoid different FDL names being converted to the same BPEL name, the tool can add a suffix to the name. • **true** Report an error in case of a possible name conflict. • **false (default)** Append a name suffix in order to solve a name conflict.
-pi[31]	<Boolean>	Initialize predefined data members. • **true** Create "assign" activities that initialize the predefined data members "_ACTIVITY" and "_PROCESS_MODEL". • **false (default)** Do not initialize the predefined data members
-pn[31]	<Boolean>	Do not create predefined data members in BPEL. • **true** Create XML schema definitions for the translated data containers without predefined data members. • **false (default)** Create XML schema definitions for the translated data containers such that they include predefined data members.
-tx	<namespace URI>	Target namespace URI for the generated XML schema file with default: **"http://www.ibm.com/xmlns/prod/websphere/mqwf/schema/"**
-tw	<namespace URI>	Target namespace URI for the generated WSDL file with default: **"http://www.ibm.com/xmlns/prod/websphere/mqwf/wsdl/"**
-tb	<namespace URI>	Target namespace URI for the generated BPEL file(s) with default: **"http://www.ibm.com/xmlns/prod/websphere/mqwf/bpel/"**
-v	<version>	Version of IBM WebSphere Process Server. Allowed values are "**v51**"[32] , "**v60**"[32], **"v602", "v61" (default)**.

Enter, for example, "**fdl2bpel -i CreditRequest.fdl**" to run the migration of a WMQWF business process that you previously exported to a file "CreditRequest.fdl":

[30] Versions "v51" and "v60" only (cp. option "-v")

[31] Note that the options "-pi" and "-pn" cannot be used together.

[32] No longer recommended.



```
D:\FDL2BPEL_TEST\Credit_Request>fdl2bpel -i CreditRequest.fdl

D:\FDL2BPEL_TEST\Credit_Request>"D:\WBIX\o0739.04\java\bin\java" -classpath "D:\BPC\Migration\FDL2BPEL-Converter\To
com.ibm.wbiserver.migration.CommonMigrationInterface.jar;D:\BPC\Migration\FDL2BPEL-Converter\Tools\Version6.1\bpecd
CWWBM0066I: FDL2BPEL converter version 6.1.
CWWBM0007I: Reading FDL input file "D:\FDL2BPEL_TEST\Credit_Request\CreditRequest.fdl".
CWWBM0173I: CreditRequest.fdl(12:18): Converting the structure name "Default Data Structure" in FDL to the structur
CWWBM0056I: CreditRequest.fdl(107:9): The FDL name "CreditRequest" cannot be used because "CreditRequest" is a rese
CWWBM0160W: CreditRequest.fdl(107:9): To avoid a name clash, the process name "CreditRequest" in FDL is converted t
CWWBM0016I: Writing output file "D:\FDL2BPEL_TEST\Credit_Request\CreditRequest.xsd".
CWWBM0016I: Writing output file "D:\FDL2BPEL_TEST\Credit_Request\CreditRequest.wsdl".
CWWBM0039W: CreditRequest.fdl(107 et seqq.): The FDL attribute "valid from" was missing in process "CreditRequest"
CWWBM0020W: CreditRequest.fdl(107 et seqq.): No BPEL mapping is available for FDL option "Keep finished processes f
CWWBM0016I: Writing output file "CreditRequest01_AcceptCredit.itel".
CWWBM0016I: Writing output file "CreditRequest01_AssessRisk.itel".
CWWBM0016I: Writing output file "CreditRequest01_CollectCreditInformation.itel".
CWWBM0016I: Writing output file "CreditRequest01_RejectCredit.itel".
CWWBM0016I: Writing output file "CreditRequest01_RequestApproval.itel".
CWWBM0021W: CreditRequest.fdl(117 et seqq.): Cannot decide whether program activity "AcceptCredit" should be mapped
tivity" ("human task (staff) activity" assumed).
CWWBM0021W: CreditRequest.fdl(127 et seqq.): Cannot decide whether program activity "AssessRisk" should be mapped t
vity" ("human task (staff) activity" assumed).
CWWBM0021W: CreditRequest.fdl(137 et seqq.): Cannot decide whether program activity "CollectCreditInformation" shou
nvocation activity" ("human task (staff) activity" assumed).
CWWBM0021W: CreditRequest.fdl(147 et seqq.): Cannot decide whether program activity "RejectCredit" should be mapped
tivity" ("human task (staff) activity" assumed).
CWWBM0021W: CreditRequest.fdl(157 et seqq.): Cannot decide whether program activity "RequestApproval" should be map
 activity" ("human task (staff) activity" assumed).
CWWBM0075W: CreditRequest.fdl(107 et seqq.): There is no data connector in process "CreditRequest" that collects th
CWWBM0016I: Writing output file "CreditRequest01_bpel.mon".
CWWBM0016I: Writing output file "D:\FDL2BPEL_TEST\Credit_Request\CreditRequest01.bpel".
CWWBM0016I: Writing output file "D:\FDL2BPEL_TEST\Credit_Request\CreditRequest01.bpelex".
CWWBM0016I: Writing output file "CreditRequest01.component".
CWWBM0005I: Exit with return code 2.

D:\FDL2BPEL_TEST\Credit_Request>_
```

Figure 12: Running FDL2BPEL from the command line

You can find more details on the generated files in "Chapter 3: Understanding the generated files" on page 32.

The FDL2BPEL Conversion Tool has the following return codes:

0	Processing finished successfully without errors or warnings.
2	Processing finished successfully and warnings occurred.
4	Errors occurred.

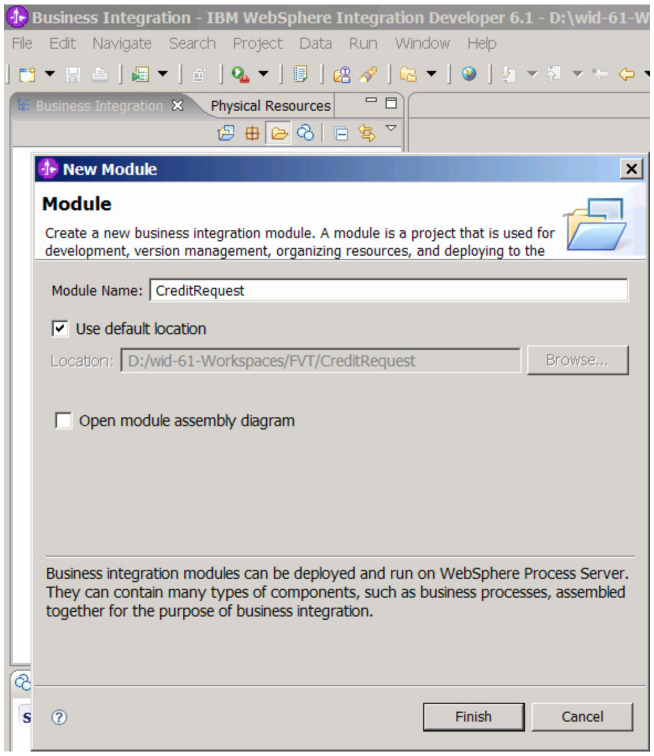
Importing the artifacts into WID

To import the generated files into WID, perform the following:

1. Start the WID.
2. Open the "Business Integration" perspective, and create a new module by clicking: **File -> New -> Module**[33]
3. Enter the desired module name, and click **Finish**.

[33] For versions "v51" and "v60" only:

Make sure that the module name you enter is the same that you specified when you ran the FDL2BPEL converter (command line tool parameter **-oe**).


Business Integration - IBM WebSphere Integration Developer 6.1 - D:\wid-61-W
File Edit Navigate Search Project Data Run Window Help
Business Integration
Physical Resources
New Module

Module

Create a new business integration module. A module is a project that is used for development, version management, organizing resources, and deploying to the

Module Name: CreditRequest

[✓] Use default location

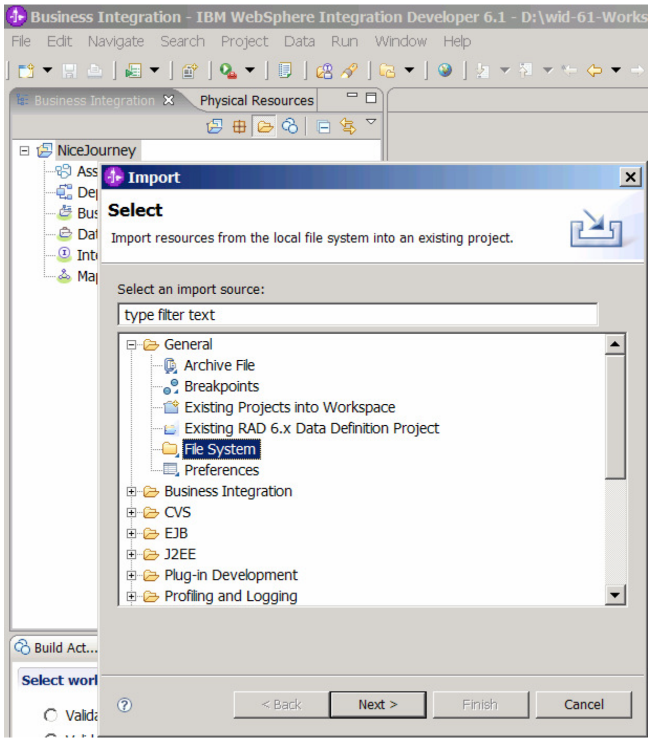
Location: D:/wid-61-Workspaces/FVT/CreditRequest

[ ] Open module assembly diagram

Business integration modules can be deployed and run on WebSphere Process Server. They can contain many types of components, such as business processes, assembled together for the purpose of business integration.

Figure 13: Creating a new business integration module

4. Select the created module, and click: **File -> Import… -> General -> File system -> Next >**.


Business Integration - IBM WebSphere Integration Developer 6.1 - D:\wid-61-Works
File Edit Navigate Search Project Data Run Window Help
Business Integration
Physical Resources
NiceJourney
Ass
Dep
Bus
Dat
Int
Map
Import
Build Act...
Select worl
Valid

Select

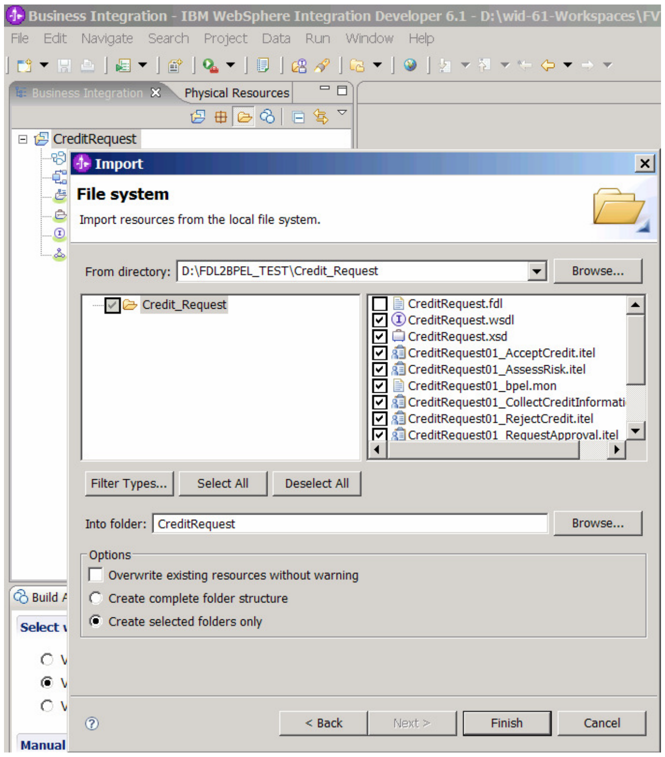
Import resources from the local file system into an existing project.

Select an import source:

type filter text

Figure 14: Importing FDL2BPEL generated files (1)

5. Browse for the source directory that contains the files that were generated by the FDL2BPEL Conversion Tool. Check the listed directory on the left side. Uncheck those files on the right side that may not belong to the files generated by the FDL2BPEL Conversion Tool (for example, the FDL file). Then click **Finish**.


Business Integration - IBM WebSphere Integration Developer 6.1 - D:\wid-61-Workspaces\FV

Import

File system

Import resources from the local file system.

From directory: D:\FDL2BPEL_TEST\Credit_Request

- [✓] Credit_Request

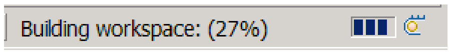
- [ ] CreditRequest.fdl
- [✓] CreditRequest.wsdl
- [✓] CreditRequest.xsd
- [✓] CreditRequest01_AcceptCredit.itel
- [✓] CreditRequest01_AssessRisk.itel
- [✓] CreditRequest01_bpel.mon
- [✓] CreditRequest01_CollectCreditInformati
- [✓] CreditRequest01_RejectCredit.itel
- [✓] CreditRequest01_RequestApproval.itel

Into folder: CreditRequest

Options
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Figure 15: Importing FDL2BPEL generated files (2)

6. Wait for the WID "Building workspace" progress indicator to reach 100%, which means you have successfully created the project module.


Building workspace: (27%)

Chapter 3: Understanding the generated files

Introduction

The FDL2BPEL Conversion Tool generates the following file types:

Table 3: Files generated by FDL2BPEL

Generated file type	File naming	File content
XSD	<FDL file name>.xsd	XML schema definitions of FDL data structures
WSDL	<FDL file name>.wsdl	Interface, binding, and address information of Web services
BPEL	<FDL process name>.bpel	XML description of a single business process model using the BPEL standard and the IBM BPEL extension definitions
BPELEX[34] [35]	<FDL process name>.bpelex	Partial XML description of the graphical layout of a single business process model
TEL[36]	<FDL activity name>.itel	XML description of tasks for people participating in the execution of a business process
COMPONENT[37]	<FDL process name>.component	XML description of an SCA process component
IMPORT[37]	<system name>_<server name>.import	XML description of an SCA MQ JMS Import (required for a UPES invocation)
JAVA[37]	<FDL data structure name> _UPES_OUT_MSG _DataBindingImpl.java	Data binding for SCA MQ JMS Import (required for a UPES invocation)
MON[38]	<FDL process name>_bpel.mon	Definitions for monitoring events

The XSD file

For each FDL file, the FDL2BPEL Conversion Tool generates a single XSD file. The XML schema definitions represent the FDL data structures in the FDL file.

The XSD file also contains XML schema definitions for the input or output data containers of the FDL constructs PROCESS, PROGRAM, BLOCK, PROGRAM_ACTIVITY, or PROCESS_ACTIVITY. Note that a data container consists of the data members of a related FDL STRUCTURE definition and additional predefined data members.

For business processes that do not exploit the predefined data members, the user can suppress their generation using the command line option **-pn** ("do not generate predefined data members")[39] or uncheck the Migration Wizard option "Create predefined data members"[40].

The XML Schema definitions are imported into the generated WSDL file.

[34] The generated BPELEX files contain the required XML definitions that control the status "collapsed" of the BPEL structured activities.

[35] Previous versions of FDL2BPEL generated also a BPELEX file with a different content (see the documentation for V5.1 and V6.0).

[36] Version "v60" or later.

[37] Version "v602" or later.

[38] Version "v61" only.

[39] See "Running the FDL2BPEL Conversion Tool" on page 27.

[40] See "Using as WID Migration Wizard" on page 21.

The WSDL file

A WSDL (Web Services Definition Language) file describes Web services in a common XML grammar. The FDL2BPEL Conversion Tool generates a single WSDL file that contains the following data, which is derived from the FDL file content:

- Data type information for all message requests and message responses
- Interface information describing all publicly available functions
- Binding information about the transport protocol to be used
- Address information for locating the specified services

The BPEL files

For each PROCESS construct found in the FDL input file, the FDL2BPEL Conversion Tool generates a single BPEL file that has the same name as the process. BPEL is used in BPC to describe business process behavior based on Web services. The language is defined as the "Business Process Execution Language for Web Services Specification, Version 1.1" (BPEL4WS[41]). Business Flow Manager (BFM) also includes extensions to the BPEL language for constructs, such as people-based activities, which are not defined in BPEL. For details, refer to the BPEL specification and the Business Flow Manager documentation.

The BPELEX files

For each process definition, a BPELEX file is generated that contains a subset of the graphical layout definitions.[42] In particular, the file contains control information about the initial "collapsed" status of BPEL structured activities.[43]

The TEL files

For each activity definition that translates to a BPEL "staff activity" (cp. section "Mapping FDL PROGRAM_ACTIVITY to BPEL"), a TEL file is generated. In WebSphere Process Server Version 6.1, the Business Process Choreographer consists of two components: BPC Business Flow Manager (BFM) and BPC Human Task Manager (HTM). The BFM implements the business process navigation capabilities available in previous releases of Business Process Choreographer, but without the human task capabilities. BPEL is used to specify the process navigation tasks. The HTM provides the human task capabilities for Business Process Choreographer. The Task Execution Language (TEL) is the corresponding XML specification language used for this purpose.

The COMPONENT files

The FDL2BPEL Conversion Tool generates an SCA component for each process definition found in the FDL file.

The IMPORT files

For each FDL user-defined program execution server (UPES) definition that is referenced by an FDL program activity so that it translates to a "Service invocation activity"[44], an SCA import file is generated. These files contain the SCA definitions for SCA imports to invoke a UPES service using an MQ JMS binding.

[41] In this documentation BPEL4WS Version 1.1 with IBM extension elements is referred to as "BPEL".

[42] WID automatically adds other layout definitions to the generated BPELEX file(s).

[43] See "Initial WID process view without fully expanded structured activities" on page 13.

[44] See "Activity classification rules" on page 60.

The JAVA files

Java files are generated for the data binding of the SCA MQ JMS import for each inbound message received from a UPES, thereby setting the message type and message type namespace.

The MON file

This file contains the definitions of business events for monitoring purposes, similar to the audit settings in WMQWF. Currently, migrating WMQWF audit event definitions to WPS is not supported. The generated .mon file just disables all event monitoring.

Chapter 4: Migration mapping rules

Mapping of FDL names to BPEL

Introduction

With the conversion of a WebSphere MQ Workflow process to WebSphere Process Server names must be checked against the syntax rules in BPEL and converted if necessary. Names of the following FDL objects are processed:

- FDL file
- Activities
- Processes
- Programs
- Servers
- Connectors
- People
- Organizations
- Roles
- Data structures
- Data structure members

For all these names the approach is the same and consists of two steps:

1. Convert the FDL name to a BPEL-compliant name, for example, by removing blanks or characters which are not allowed in BPEL names.
2. Check if this new BPEL-compliant name is already in use for another FDL concept which might cause a name conflict in the BPEL process model. If this is the case, the BPEL-compliant name is modified by appending a suitable number such that the name conflict is avoided.

Keep in mind that this naming conversion algorithm can have unwanted side-effects. For example, if you redesign your FDL process, due to a different order in the FDL file, names could be converted differently compared to a previous run of the migration tool.

This problem can be avoided if in the FDL processes you use only BPEL-compliant names. For this, consider the option **-x** of the WebSphere MQ Workflow runtime export and import utility (fmcibie). See the documentation "IBM WebSphere MQ Workflow Getting Started with Buildtime" for more information.

Name mapping rules in detail

1. Activity names are truncated if necessary so that they are not longer than 64 bytes as UTF-8 string.
2. For all names, blanks are removed.
3. For all names, "(" and ")" are replaced by "_".
4. If an activity name, member name, process name, program name, or data structure name starts with a digit, then the character "N" is prefixed before the name.
5. Member name and data structure names are converted to valid Java identifier names. If the first character of the name is not an allowed first character of a Java identifier but an allowed character of a Java identifier, the name is prefixed with a letter "N".

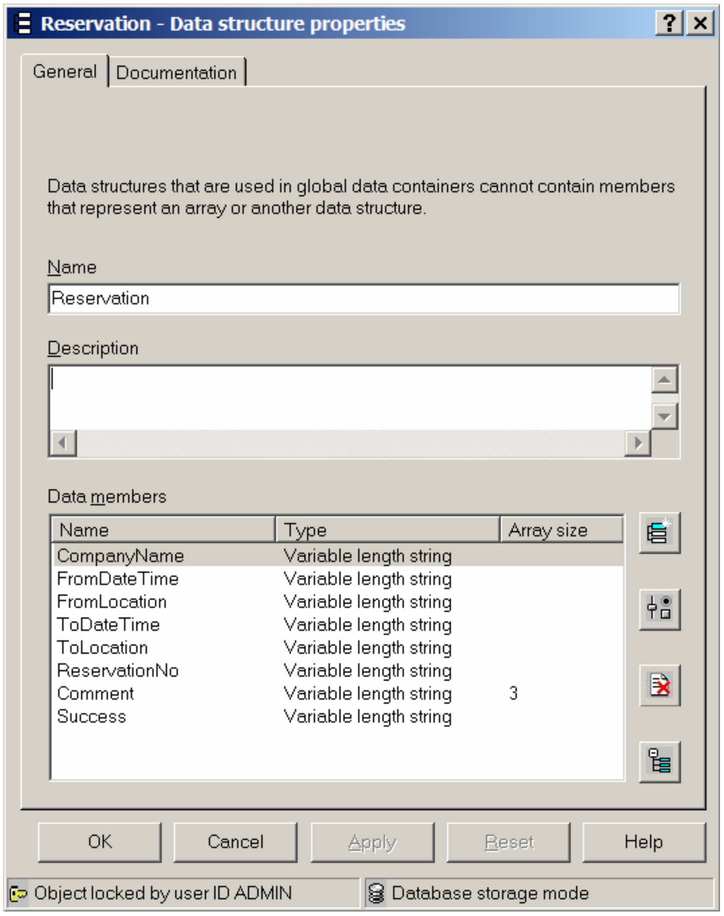
Then, all characters of the name which are not allowed characters in a Java identifier are replaced by an "_". In addition, the first character of the name is made uppercase.

6. If a member name or data structure name is "Class", then this name is replaced by a generated name.
7. File names and activity names are checked if they contain non-migratable special characters outside of the following set: a-z, A-Z, 0-9, -, ., _, ~. Non-migratable characters are replaced by "_".

FDL to XSD mapping rules

Mapping FDL "STRUCTURE" to a "Business Object"

An FDL data structure definition consists of a sequence of data members that are specified as pairs of member names and data types. For example:



Data structures that are used in global data containers cannot contain members that represent an array or another data structure.

Name: Reservation

Description:

Data members

Name	Type	Array size
CompanyName	Variable length string	
FromDateTime	Variable length string	
FromLocation	Variable length string	
ToDateTime	Variable length string	
ToLocation	Variable length string	
ReservationNo	Variable length string	
Comment	Variable length string	3
Success	Variable length string	

Object locked by user ID ADMIN
Database storage mode

Figure 16: Sample FDL data structure

If you export such a data structure definition into an FDL file you get the external FDL format:

```
STRUCTURE 'Reservation'
  'CompanyName': STRING;
  'FromDateTime': STRING;
  'FromLocation': STRING;
  'ToDateTime': STRING;
  'ToLocation': STRING;
  'ReservationNo': STRING;
  'Comment': STRING[3];
  'Success': STRING;
END 'Reservation'
```

The FDL2BPEL Conversion Tool translates that FDL data structure definition into a corresponding XML Schema definition like this:

```
<complexType name="Reservation">
   <sequence...>
      <element name="CompanyName" type="xsd:string" minOccurs="0"/>
      <element name="FromDateTime" type="xsd:string" minOccurs="0"/>
      <element name="FromLocation" type="xsd:string" minOccurs="0"/>
      <element name="ToDateTime" type="xsd:string" minOccurs="0"/>
      <element name="ToLocation" type="xsd:string" minOccurs="0"/>
      <element name="ReservationNo" type="xsd:string" minOccurs="0"/>
      <element name="Comment" type="xsd:string" minOccurs="0" maxOccurs="3"/>
      <element name="Success" type="xsd:string" minOccurs="0"/>
   </sequence>
 </complexType>
```

The basic FDL data types map onto the XML schema type system, as follows:

Table 4: Mapping basic FDL data types

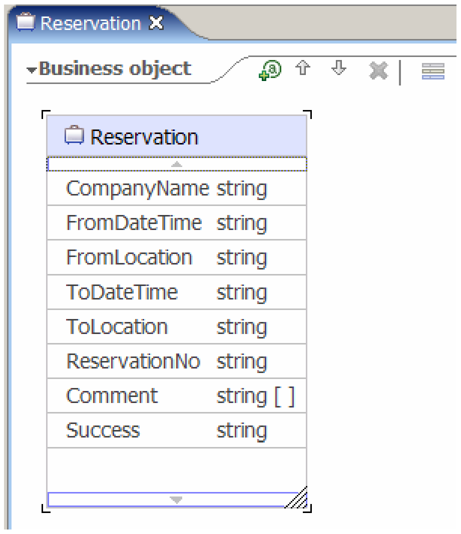
FDL data type	XML schema data type
STRING	xsd:string
LONG	xsd:long
FLOAT	xsd:double
BINARY	xsd:base64Binary

Elements are specified with a minimum occurrence of zero because WMQWF allows data structure members to be ‘not set’.

(Sparse) arrays, which is the only collection type supported by WMQWF, are mapped to elements with the corresponding maximum cardinality.[45]

The WID business object editor allows you to display the above XML schema definition in a more intuitive graphical mode:

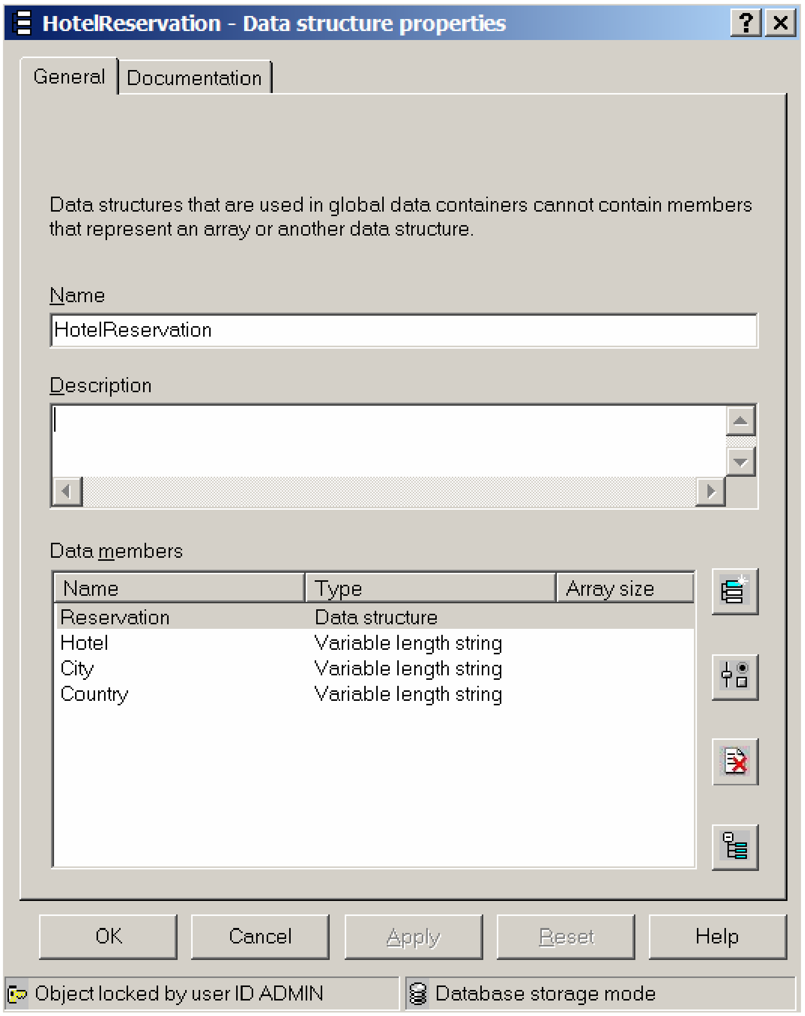
[45] See "Data arrays" on page 135.



CompanyName	string
FromDateTime	string
FromLocation	string
ToDateTime	string
ToLocation	string
ReservationNo	string
Comment	string [ ]
Success	string

Figure 17: WID Business Object Editor

Naturally, an FDL data structure can be hierarchically organized, that is, a data structure member may refer to another data structure by its type:



Data structures that are used in global data containers cannot contain members that represent an array or another data structure.

Name: HotelReservation

Description:

Data members

Name	Type	Array size
Reservation	Data structure	
Hotel	Variable length string	
City	Variable length string	
Country	Variable length string	

Object locked by user ID ADMIN

Database storage mode

Figure 18: Hierarchical FDL data structure

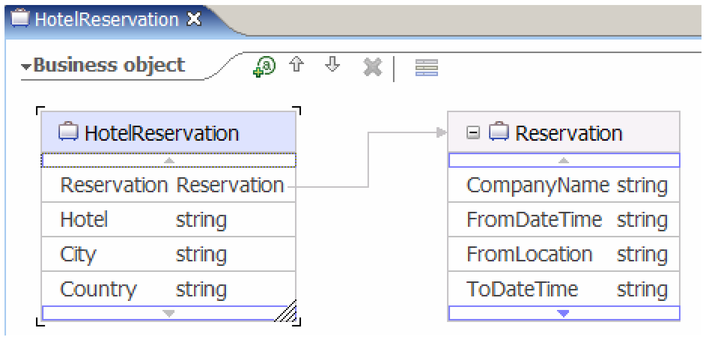
The external FDL format shows the name of the referenced "Reservation" data structure as the type of the corresponding member of "HotelReservation":

```
STRUCTURE 'HotelReservation'
  'Reservation': 'Reservation';
  Hotel: STRING;
  City: STRING;
  Country: STRING;
END 'HotelReservation'
```

Here is the corresponding XML schema definition generated by the FDL2BPEL Conversion Tool:

```
<complexType name="HotelReservation">
   <sequence...>
       <element name="Reservation" type="mqwf:Reservation" minOccurs="0"/>
       <element name="Hotel" type="xsd:string" minOccurs="0"/>
       <element name="City" type="xsd:string" minOccurs="0"/>
       <element name="Country" type="xsd:string" minOccurs="0"/>
   </sequence>
</complexType>
```

The WID Business Object editor indicates the "element of" relationship between "HotelReservation" and "Reservation" with an arrow:



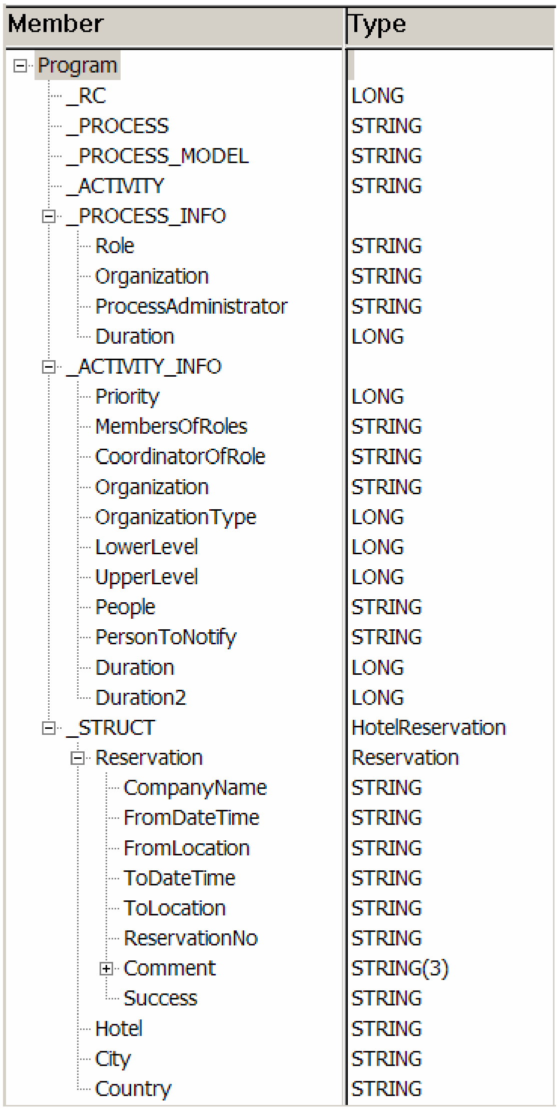
Reservation	Reservation
Hotel	string
City	string
Country	string

CompanyName	string
FromDateTime	string
FromLocation	string
ToDateTime	string

Figure 19: Hierarchically structured Business Object

Mapping an FDL data container to an XML schema definition

Data that is passed between activities in an FDL process consist of an aggregation of a user-defined data structure and predefined data members. Such data aggregation is called a "data container". For instance, the "HotelReservation" that is used as the data input or output of an activity becomes part of a data container, as follows:



Member	Type
Program	
_RC	LONG
_PROCESS	STRING
_PROCESS_MODEL	STRING
_ACTIVITY	STRING
_PROCESS_INFO	
Role	STRING
Organization	STRING
ProcessAdministrator	STRING
Duration	LONG
_ACTIVITY_INFO	
Priority	LONG
MembersOfRoles	STRING
CoordinatorOfRole	STRING
Organization	STRING
OrganizationType	LONG
LowerLevel	LONG
UpperLevel	LONG
People	STRING
PersonToNotify	STRING
Duration	LONG
Duration2	LONG
_STRUCT	HotelReservation
Reservation	Reservation
CompanyName	STRING
FromDateTime	STRING
FromLocation	STRING
ToDateTime	STRING
ToLocation	STRING
ReservationNo	STRING
Comment	STRING(3)
Success	STRING
Hotel	STRING
City	STRING
Country	STRING

Figure 20: FDL data container

A data container does not need to be defined explicitly in the FDL file. However, if you want to use the predefined data members in the converted process model, the data container must be considered by the FDL2BPEL Conversion Tool.

For instance, the above data container with the embedded data structure "HotelReservation" is mapped to the following XML schema definition:

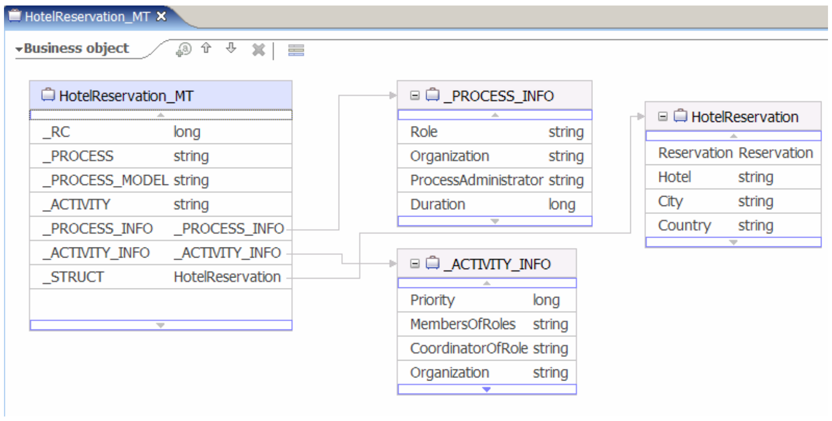
```
<complexType name="HotelReservation_MT">
    <sequence...>
        <element name="_RC" type="xsd:long" minOccurs="0"/>
        <element name="_PROCESS" type="xsd:string" minOccurs="0"/>
        <element name="_PROCESS_MODEL" type="xsd:string" minOccurs="0"/>
        <element name="_ACTIVITY" type="xsd:string" minOccurs="0"/>
        <element name="_PROCESS_INFO" type="mqwf:_PROCESS_INFO">
        <element name="_ACTIVITY_INFO" type="mqwf:_ACTIVITY_INFO"/>
        <element name="_STRUCT" type="mqwf: HotelReservation"/>
    </sequence>
</complexType>
```

The predefined data members "_PROCESS_INFO" and "_ACTIVITY_INFO" refer to other predefined data structures which must also be mapped to XML schema definitions:

```
<complexType name="_PROCESS_INFO">
   <sequence...>
      <element name="Role" type="xsd:string" minOccurs="0"/>
      <element name="Organization" type="xsd:string" minOccurs="0"/>
      <element name="ProcessAdministrator" type="xsd:string" minOccurs="0"/>
      <element name="Duration" type="xsd:long" minOccurs="0"/>
   </sequence>
</complexType>

<complexType name="_ACTIVITY_INFO">
   <sequence...>
      <element name="Priority" type="xsd:long" minOccurs="0"/>
      <element name="MembersOfRoles" type="xsd:string" minOccurs="0"/>
      <element name="CoordinatorOfRole" type="xsd:string" minOccurs="0"/>
      <element name="Organization" type="xsd:string" minOccurs="0"/>
      <element name="OrganizationType" type="xsd:long" minOccurs="0"/>
      <element name="LowerLevel" type="xsd:long" minOccurs="0"/>
      <element name="UpperLevel" type="xsd:long" minOccurs="0"/>
      <element name="People" type="xsd:string" minOccurs="0"/>
      <element name="PersonToNotify" type="xsd:string" minOccurs="0"/>
      <element name="Duration" type="xsd:long" minOccurs="0"/>
      <element name="Duration2" type="xsd:long" minOccurs="0"/>
   </sequence>
</complexType>
```

The WID Business Object editor renders the migrated data container in a graphical view:



HotelReservation_MT	
_RC	long
_PROCESS	string
_PROCESS_MODEL	string
_ACTIVITY	string
_PROCESS_INFO	_PROCESS_INFO
_ACTIVITY_INFO	_ACTIVITY_INFO
_STRUCT	HotelReservation

_PROCESS_INFO	
Role	string
Organization	string
ProcessAdministrator	string
Duration	long

_ACTIVITY_INFO	
Priority	long
MembersOfRoles	string
CoordinatorOfRole	string
Organization	string

HotelReservation	
Reservation	Reservation
Hotel	string
City	string
Country	string

Figure 21: Mapping FDL data container to Business Object

You can suppress the generation of predefined data members by using the command line option **-pn** ("do not generate predefined data members")[46] or by unchecking the Migration Wizard option "Create predefined data members"[47]. Note that this option will not completely remove the generation of the above data structures "_PROCESS_INFO" and "_ACTIVITY_INFO" because those are still needed as members of data structure "_MESSAGE_CONTEXT" (see section "Mapping the data exchanged with a UPES").

Mapping the data exchanged with a UPES

WMQWF supports the notion of a UPES, which makes it possible to send program invocation request messages in XML format to a user-defined MQ Workflow queue (see the documentation "IBM WebSphere MQ Workflow Programming Guide"). The FDL2BPEL Conversion Tool supports reusing a UPES in a business process that you converted to BPEL. A UPES requires the specification of another message type that can be passed as input to or output from

[46] See "Running the FDL2BPEL Conversion Tool" on page 27.

[47] See "Using as WID Migration Wizard" on page 21.

a corresponding "UPES Web service" in a WPS environment. This UPES message type consists of "container" data and "context" data. The "container" data consists of the original container data. The "context" data requires the generation of another XML schema definition in the XSD file:

```
<complexType name="_MESSAGE_CONTEXT">
    <sequence...>
        <element name="_RC" type="xsd:long" minOccurs="0"/>
        <element name="_PROCESS" type="xsd:string" minOccurs="0"/>
        <element name="_PROCESS_MODEL" type="xsd:string" minOccurs="0"/>
        <element name="_ACTIVITY" type="xsd:string" minOccurs="0"/>
        <element name="_PROCESS_INFO" type="mqwf:_PROCESS_INFO" minOccurs="0"/>
        <element name="_ACTIVITY_INFO" type="mqwf:_ACTIVITY_INFO" minOccurs="0"/>
        <element name="ResponseRequired" type="xsd:string" minOccurs="0"/>
        <element name="UserContext" type="xsd:string" minOccurs="0"/>
        <element name="KeepName" type="xsd:string" minOccurs="0"/>
        <element name="ActImplCorrelID" type="xsd:string" minOccurs="0"/>
        <element name="Starter" type="xsd:string" minOccurs="0"/>
        <element name="ProgramName" type="xsd:string" minOccurs="0"/>
        <element name="MessageType" type="xsd:string" minOccurs="0"/>
        <element name="Origin" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstID" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstParentName" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstTopLevelName" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstDescription" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstState" type="xsd:string" minOccurs="0"/>
        <element name="LastStateChangeTime" type="xsd:string" minOccurs="0"/>
        <element name="LastModificationTime" type="xsd:string" minOccurs="0"/>
        <element name="ProcTemplID" type="xsd:string" minOccurs="0"/>
        <element name="ProcTemplValidFromDate" type="xsd:string" minOccurs="0"/>
        <element name="Icon" type="xsd:string" minOccurs="0"/>
        <element name="Category" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstSuspensionTime" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstSuspensionExpirationTime" type="xsd:string" minOccurs="0"/>
        <element name="Rc" type="xsd:long" minOccurs="0"/>
        <element name="MessageText" type="xsd:string" minOccurs="0"/>
        <element name="ProcTemplName" type="xsd:string" minOccurs="0"/>
        <element name="ProcInstName" type="xsd:string" minOccurs="0"/>
        <element name="ProgramRC" type="xsd:long" minOccurs="0"/>
        <element name="ExternalProcessContext" type="xsd:string" minOccurs="0"/>
    </sequence>
</complexType>[48]
```

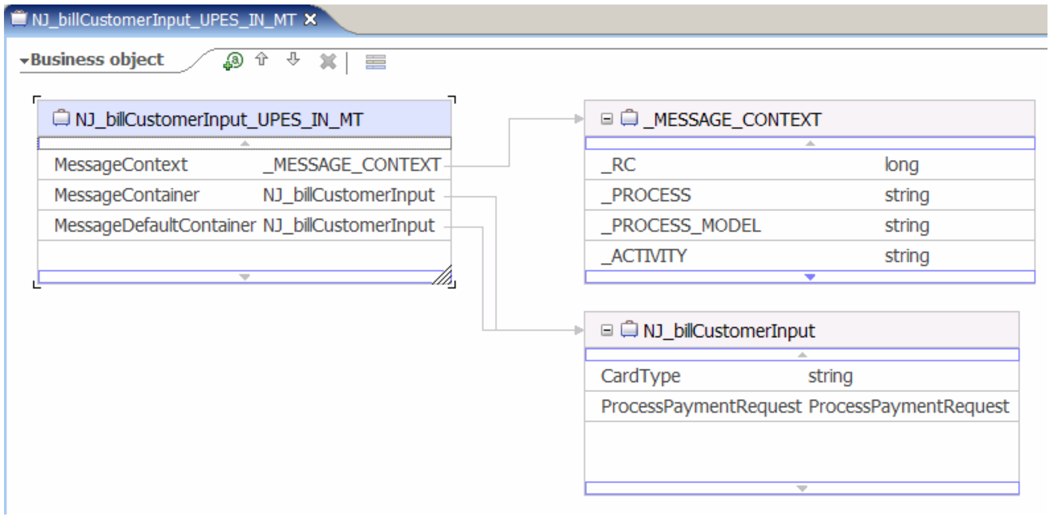
The migration tool creates the messages that represent FDL container data as well as a "wrapper" containing the data itself and the context (see above).

Here is an example of how a message called "NJ_billCustomerInput" would be displayed by a text editor and by the WID Business Object editor:

[48] Note**:** The element "ImplementationData" and its sub elements, which WMQWF supports for WMQWF XML message type "ActivityImplInvoke" is not supported and therefore is not part of the generated _MESSAGE_CONTEXT structure (cp. limitation "XML message types" on page 140).

UPES input message:

```
<complexType name="NJ_billCustomerInput_UPES_IN_MT">
  <sequence...>
    <element name="MessageContext" type="mqwf:_MESSAGE_CONTEXT"/>
    <element name="MessageContainer" type="mqwf:NJ_billCustomerInput"/>
    <element name="MessageDefaultContainer" type="mqwf:NJ_billCustomerInput"/>
  </sequence>
</complexType>
```



NJ_billCustomerInput_UPES_IN_MT

MessageContext	_MESSAGE_CONTEXT
MessageContainer	NJ_billCustomerInput
MessageDefaultContainer	NJ_billCustomerInput

_MESSAGE_CONTEXT

_RC	long
_PROCESS	string
_PROCESS_MODEL	string
_ACTIVITY	string

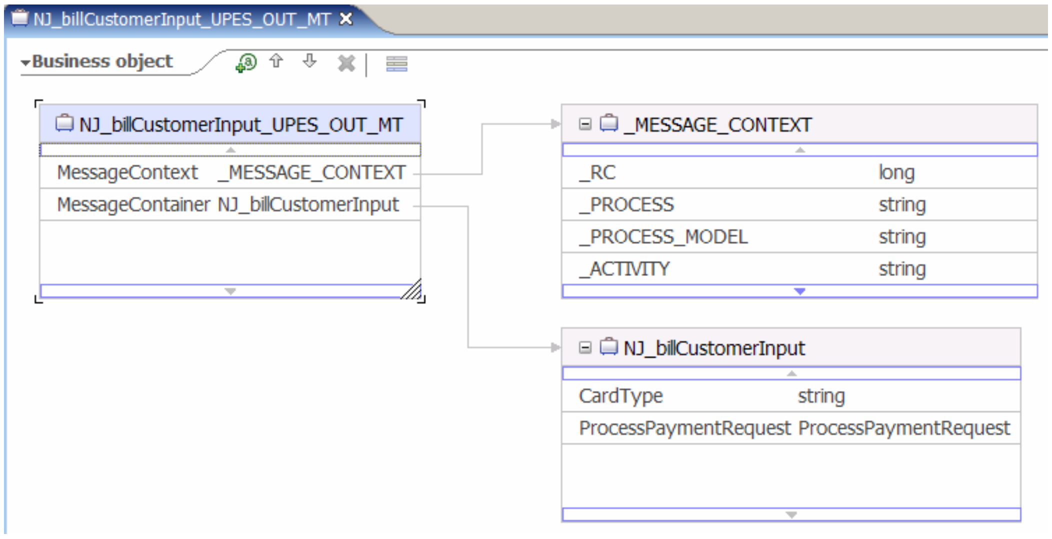
NJ_billCustomerInput

CardType	string
ProcessPaymentRequest	ProcessPaymentRequest

Figure 22: UPES input message as Business Object

UPES output message:

```
<complexType name="NJ_billCustomerInput_UPES_OUT_MT">
  <sequence...>
    <element name="MessageContext" type="mqwf:_MESSAGE_CONTEXT"/>
    <element name="MessageContainer" type="mqwf:NJ_billCustomerInput"/>
  </sequence>
</complexType>
```



NJ_billCustomerInput_UPES_OUT_MT

MessageContext	_MESSAGE_CONTEXT
MessageContainer	NJ_billCustomerInput

_MESSAGE_CONTEXT

_RC	long
_PROCESS	string
_PROCESS_MODEL	string
_ACTIVITY	string

NJ_billCustomerInput

CardType	string
ProcessPaymentRequest	ProcessPaymentRequest

Figure 23: UPES output message as Business Object

FDL to WSDL mapping rules

In the BPEL model, interactions of a business process with the ‘outside world’ (as the caller and as the called) are based on Web services, which are described in WSDL.

Generation of partner link type definitions

A BPEL partner link type is introduced as an extension element in the generated WSDL file. It characterizes the conversational relationship between two services by defining the "roles" played by each of the services in the conversation. It also specifies the "port type" (or "interface") provided by each service to receive messages within the context of the conversation. For more details, refer to the BPEL specification.

Migrating a MQ Workflow process requires that two categories of partner link types are generated:

- Partner link type that represents a business process
- Partner link type that represents a UPES as a Web service

Mapping FDL PROCESS to a partner link type

For each FDL PROCESS construct in the FDL input file, the FDL2BPEL Conversion Tool creates a partner link definition. For example[49]:

```
<!-- Partner link type for process "Nice_Journey" -->
<plnk:partnerLinkType name="Nice_Journey_PLT">
   <plnk:role name="Nice_Journey_Role">
      <plnk:portType name="wsdl1:Nice_Journey_PT"/>
   </plnk:role>
</plnk:partnerLinkType>
```

The "plnk:portType" element refers to a "port type" definition that describes the interaction protocol (interface) of the respective role.

If the "Nice_Journey" process has a subprocess "NJ_BookCar" then a second partner link type is added:

[49] Note that there is no graphic view of a partner link type in WID available, this is because you never need to create or modify a partner link type manually.

```
<!-- Partner link type for process "NJ_BookCar" -->
<plnk:partnerLinkType name="NJ_BookCar_PLT">
    <plnk:role name="NJ_BookCar_Role">
        <plnk:portType name="wsdl1:NJ_BookCar_PT"/>
    </plnk:role>
</plnk:partnerLinkType>
```

Mapping FDL SERVER to a BPEL partner link type

For every FDL SERVER definition that represents a UPES the FDL2BPEL Conversion Tool creates a partner link type. For example:

```
<!-- Partner link type for program "NJ_BookHotel" -->
<plnk:partnerLinkType name="NJ_BookHotel_PLT">
    <plnk:role name="NJ_BookHotel_Role">
        <plnk:portType name="wsdl1:NJ_BookHotel_PT"/>
    </plnk:role>
</plnk:partnerLinkType>
```

Mapping FDL container data to WSDL message definitions

A WSDL "message" definition represents a business object that is used in a business process as a message. For instance, a business object that is input to an activity is used as a "message" to request some kind of service. The business object returned from the activity as its output data represents the corresponding response "message". A WSDL message definition consists of a name attribute and zero or more message "part" elements.

The FDL2BPEL Conversion Tool creates messages that represent FDL container data[50] having only a single "part" element. For example:

```
<!-- Request message of program activity "ReserveFlight" -->
<message name="ReserveFlight_Request_MSG">
     <part name="Part1" element="wsdl1:ReserveFlight"/>
</message>
```

The type attribute of the "part" element refers to an XML schema element definition (here: "ReserveFlight") in the "types" section of the WSDL. This one, in turn, refers to an XML schema definition of the respective FDL data container (here: "makeReservation_MT") that you find in the generated XSD file:

[50] See "Mapping an FDL data container to an XML schema definition" on page 39.

```
<types>
    <xsd:schema targetNamespace="http://www.ibm.com/xmlns/prod/websphere/mqwf/wsdl/">
        <xsd:import schemaLocation="NiceJourney.xsd"
                    namespace="http://www.ibm.com/xmlns/prod/websphere/mqwf/schema/"/>
        <!-- *** XMLSchema elements *** -->
        ...
        <!-- XMLSchema element belonging to request message of program activity "ReserveFlight" -->
        <xsd:element name="ReserveFlight">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="input1" type="mqwf:makeReservation_MT"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        ....
    </xsd:schema>
</types>
```

The FDL2BPEL Conversion Tool also generates a special message element with the name "ActivityStatus_MSG". It is used to map FDL exit conditions to the "condition" attribute of a BPEL <while> activity (cp. section "Exit conditions" on page 73):

```
<message name="ActivityStatus_MSG">
  <part name="Part1" type="xsd:string"/>
</message>
```

Mapping FDL interaction interfaces to WSDL port types

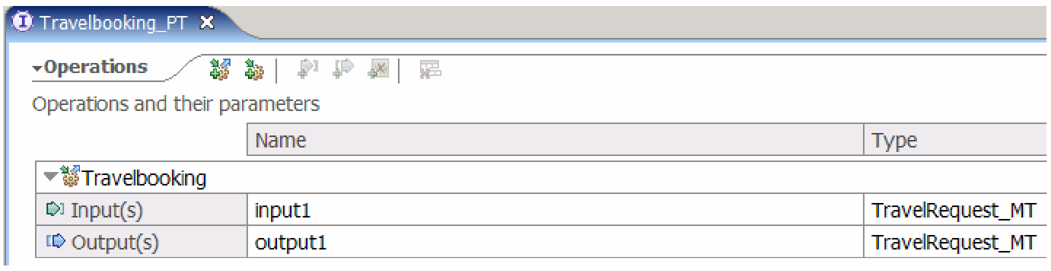
The WSDL "portType" element combines multiple message elements to form a complete one-way or a request-response operation. A WSDL "portType" element can define multiple operations.

Mapping FDL PROCESS to a port type

The FDL2BPEL Conversion Tool maps the interface of an FDL PROCESS construct to a "portType" element with a single request/response operation. For example:

```
<!-- Port type for process "Travelbooking" -->
<portType name="Travelbooking_PT">
  <operation name="Travelbooking">
    <input name="Travelbooking_Request" message="wsdl1: Travelbooking_Request_MSG"/>
    <output name="Travelbooking_Response" message="wsdl1: Travelbooking_Response_MSG"/>
  </operation>
</portType>
```

The WID Interface editor provides the following graphical view of the same port type:



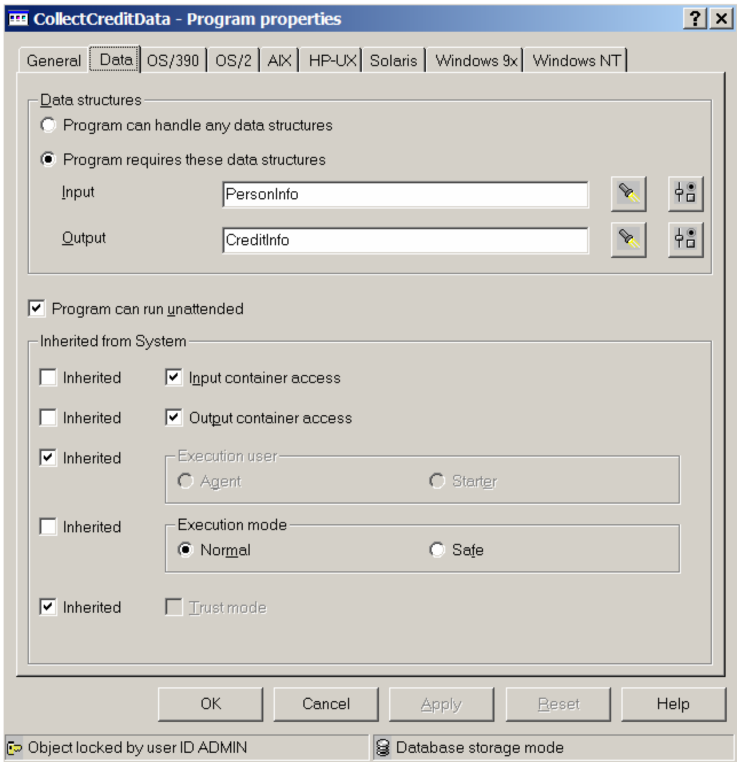
	Name	Type
Travelbooking		
Input(s)	input1	TravelRequest_MT
Output(s)	output1	TravelRequest_MT

Figure 24: Process interface

Mapping FDL PROGRAM to a port type

WMQWF describes the conversational relationship or interaction of a business process with another application service by the abstract notion of a "PROGRAM". The mapping to a port type depends on which data a program can handle:

If the program property "***Program requires these data structures***" is enabled, the generated port type will usually have a single "operation" element.



Data structures
- ( ) Program can handle any data structures
- (●) Program requires these data structures

Input: PersonInfo
Output: CreditInfo

[✓] Program can run unattended

Inherited from System
- [ ] Inherited — [✓] Input container access
- [ ] Inherited — [✓] Output container access
- [✓] Inherited — Execution user: ( ) Agent ( ) Starter
- [ ] Inherited — Execution mode: (●) Normal ( ) Safe
- [✓] Inherited — [ ] Trust mode

Object locked by user ID ADMIN | Database storage mode

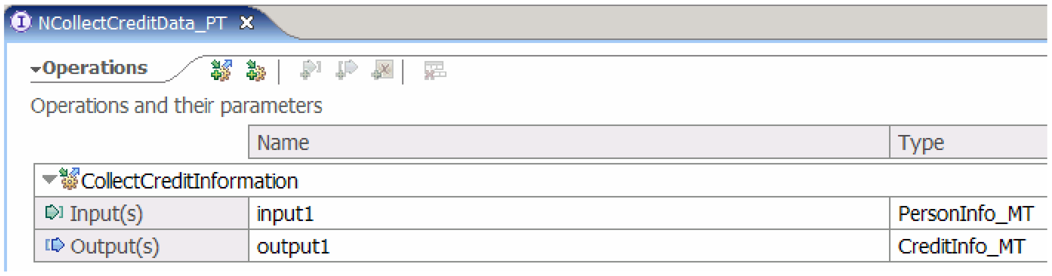
Figure 25: Program requires these data structures

For instance, a program "NCollectCreditData" translates to a port type "NCollectCreditData _PT", as follows:

WSDL source code:

```
<!-- Port type for program "NCollectCreditData" -->
<portType name="NCollectCreditData_PT">
    <!-- Operation used in activity "CollectCreditInformation" -->
    <operation name="CollectCreditInformation">
        <input name="CollectCreditInformation_Request" message="wsdl1:CollectCreditInformation_Request_MSG"/>
        <output name="CollectCreditInformation_Response" message="wsdl1:CollectCreditInformation_Response_MSG"/>
    </operation>
</portType>
```

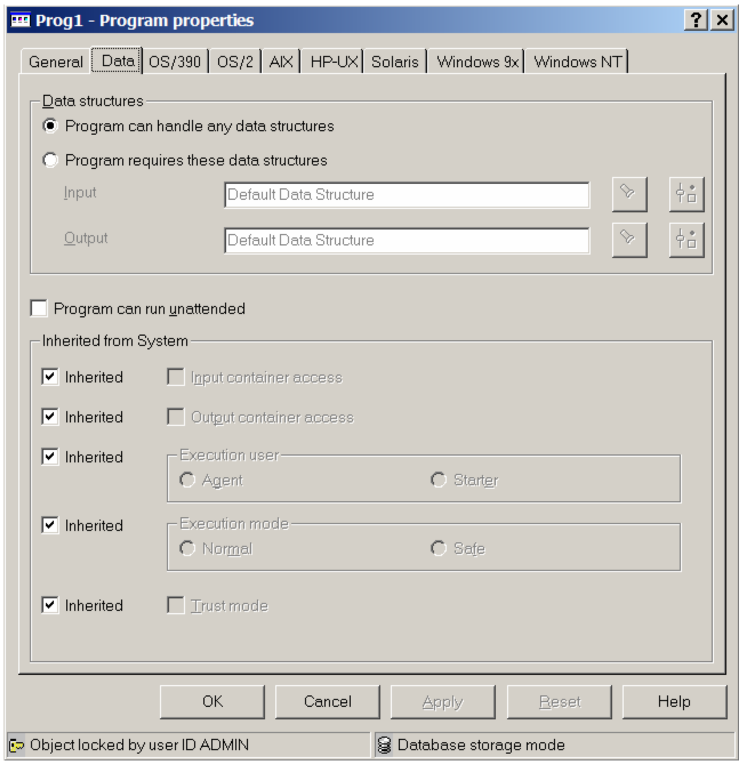
WID interface editor:



	Name	Type
CollectCreditInformation		
Input(s)	input1	PersonInfo_MT
Output(s)	output1	CreditInfo_MT

Figure 26: Interface with single operation

If the program property "***Program can handle any data structures***" is enabled for a program "Prog1", the number of "operation" elements in the generated port type depends on the number of FDL PROGRAM_ACTIVITY constructs that specify program "Prog1".



Data structures
(● Program can handle any data structures) ○ Program requires these data structures
Input: Default Data Structure
Output: Default Data Structure

☐ Program can run unattended

Inherited from System
☑ Inherited ☐ Input container access
☑ Inherited ☐ Output container access
☑ Inherited — Execution user: ○ Agent ○ Starter
☑ Inherited — Execution mode: ○ Normal ○ Safe
☑ Inherited ☐ Trust mode

Object locked by user ID ADMIN | Database storage mode

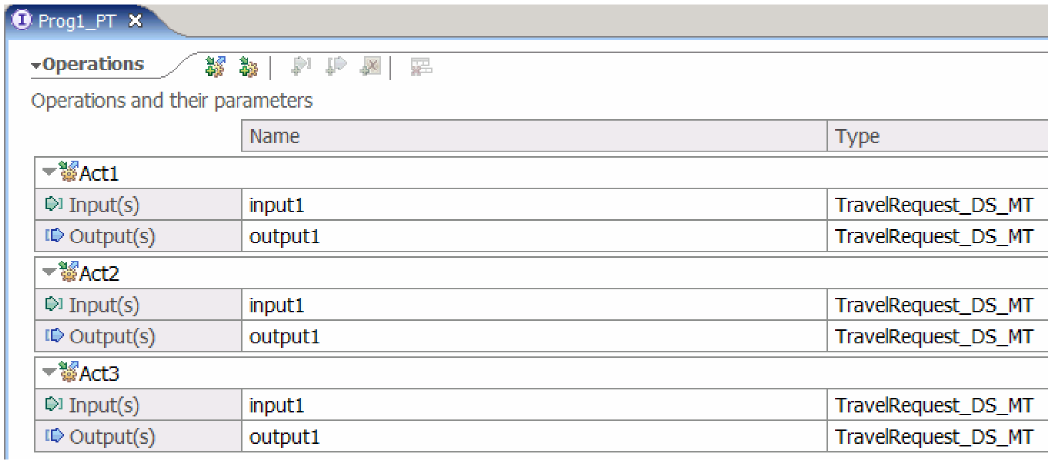
Figure 27: Program can handle any data structures

For instance, if activities "Act1", Act2", "Act3" specify program "Prog1", the FDL2BPEL Conversion Tool will generate a port type with three operations:

WSDL source code:

```
<!-- Port type for program "Prog1" -->
<portType name="Prog1_PT">
  <!-- Operation used in activity "Act1" -->
  <operation name="Act1">
    <input name="Act1_Request" message="wsdl1: Act1_Request _MSG"/>
    <output name="Act1_Response" message="wsdl1: Act1_Response_MSG"/>
  </operation>
  <!-- Operation used in activity "Act2" -->
  <operation name="Act2">
    <input name="Act2_Request" message="wsdl1: Act2_Request _MSG"/>
    <output name="Act2_Response" message="wsdl1: Act2_Response_MSG"/>
  </operation>
  <!-- Operation used in activity "Act3" -->
  <operation name="Act3">
    <input name="Act3_Request" message="wsdl1: Act2_Request _MSG"/>
    <output name="Act3_Response" message="wsdl1: Act3_Response_MSG"/>
  </operation>
</portType>
```

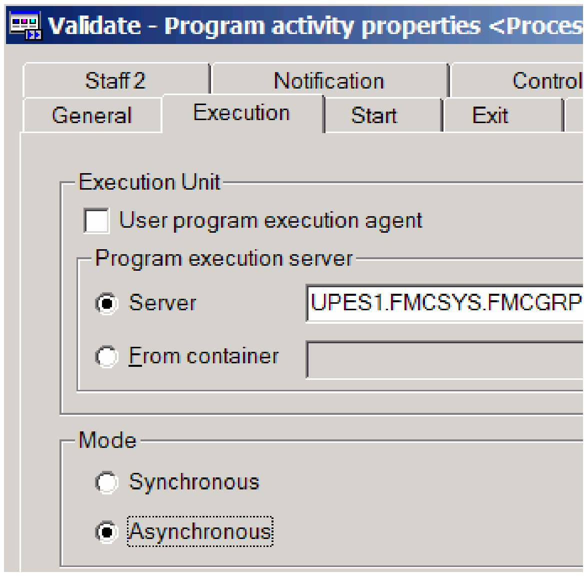
WID interface editor:



	Name	Type
Act1		
Input(s)	input1	TravelRequest_DS_MT
Output(s)	output1	TravelRequest_DS_MT
Act2		
Input(s)	input1	TravelRequest_DS_MT
Output(s)	output1	TravelRequest_DS_MT
Act3		
Input(s)	input1	TravelRequest_DS_MT
Output(s)	output1	TravelRequest_DS_MT

Figure 28: Interface with multiple operations

The above <operation…> elements of a port type may occur without an <output…> element (which, in WSDL terminology, denotes a one-way operation). This means that it is an "asynchronous" operation where the control flow continues after sending the input message without waiting for an output message. "Asynchronous" operations are generated for "UPES" activities defined as "asynchronous" in WMQWF Buildtime. Example:


Validate - Program activity properties <Proces

Tabs: Staff 2, Notification, Control, General, Execution, Start, Exit

Execution Unit
- [ ] User program execution agent

Program execution server
- (●) Server: UPES1.FMCSYS.FMCGRP
- ( ) From container:

Mode
- ( ) Synchronous
- (●) Asynchronous

Figure 29: Asynchronous execution of UPES

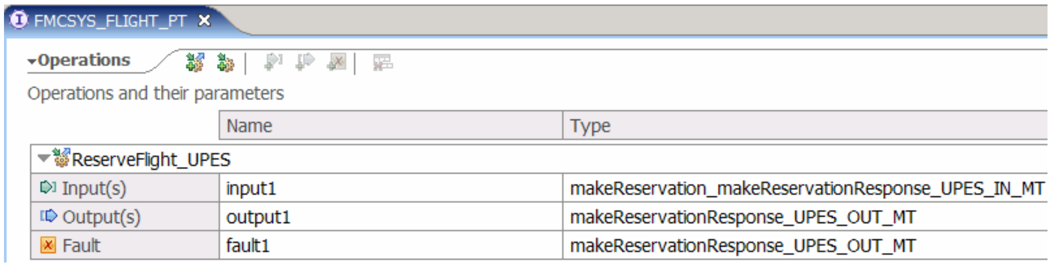
Mapping FDL SERVER to a port type for a UPES

As described in the section "Mapping the data exchanged with a UPES" on page 41, a UPES requires special message types that can be exchanged with a corresponding "UPES Web service". Another WSDL port type definition is used to describe the interface. For example:

WSDL source code:

```
<!-- Port type for server "FLIGHT" of system "FMCSYS" -->
<portType name="FMCSYS_FLIGHT_PT">
	<!-- Operation used in activity "ReserveFlight" -->
	<operation name="ReserveFlight_UPES">
		<input name="ReserveFlight_Request_UPES" message="wsdl1:ReserveFlight_Request_UPES_MSG"/>
		<output name="ReserveFlight_Response_UPES" message="wsdl1:ReserveFlight_Response_UPES_MSG"/>
		<fault name="fault1" message="wsdl1:ReserveFlight_Fault_UPES_MSG"/>
	</operation>
</portType>
```

WID interface editor:


FMCSYS_FLIGHT_PT

Operations

Operations and their parameters

	Name	Type
ReserveFlight_UPES		
Input(s)	input1	makeReservation_makeReservationResponse_UPES_IN_MT
Output(s)	output1	makeReservationResponse_UPES_OUT_MT
Fault	fault1	makeReservationResponse_UPES_OUT_MT

Figure 30: UPES interface

The port type shown above contains an operation of type "request-response", which is required for a "synchronous" UPES invocation. Note that the "request-response" operation now has another "fault" element that is needed for any exception returned from the UPES. You must decide how to respond to a "fault message" in your business logic. For example, you can add a "fault handler" to the generated BPEL process. Because FDL has no concept for a "fault handler", the FDL2BPEL Conversion Tool cannot automatically derive this from the FDL input file.

FDL to BPEL mapping rules

Mapping FDL PROCESS to BPEL

An FDL process is mapped to a corresponding BPEL process: For each "PROCESS" found in the FDL file, the FDL2BPEL Conversion Tool creates a separate file with the extension ".bpel". The file name is the same as the process name. If necessary, the process name is converted to a valid BPEL name.

The BPEL file contains <process> as the root element. Here is an example of the corresponding properties that are generated:

BPEL source code:

```
<bpws:process … name="Nice_Journey" wpc:displayName=" Nice_Journey" … >
```

You can review these properties in WID as follows:

1. Open the BPEL process in the Business Integration view with the business process editor:

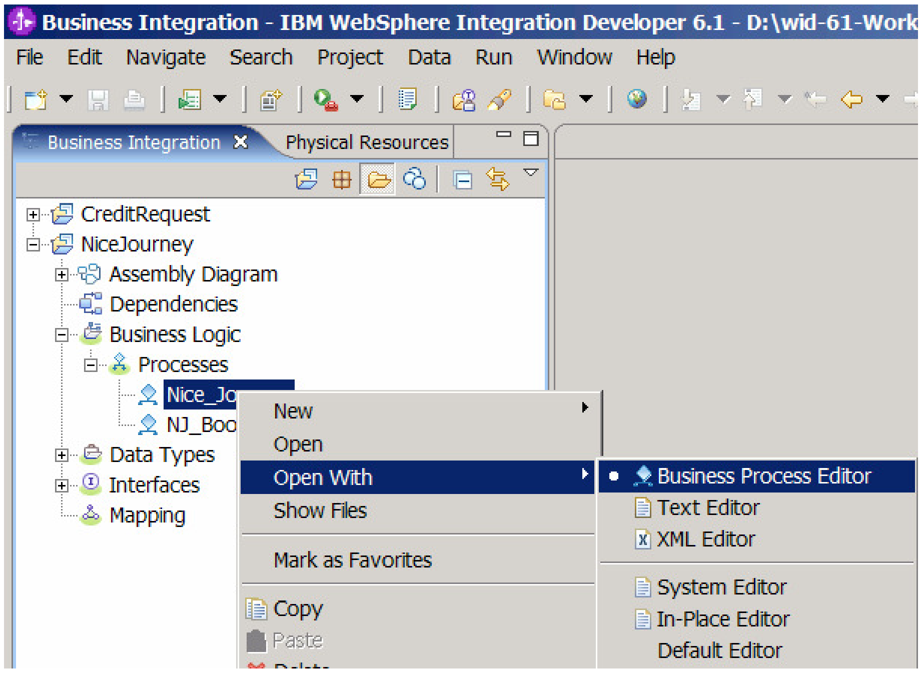
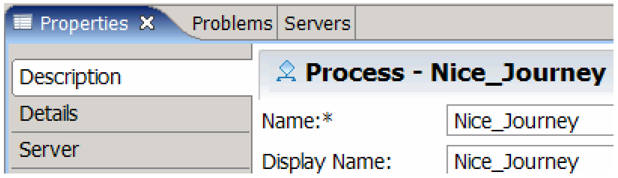


Figure 31: Opening the business process editor

2. Select the Description page of the Properties view:



Name:* Nice_Journey
Display Name: Nice_Journey

Figure 32: Mapping process properties (1)

Further examples of process properties are:[51]

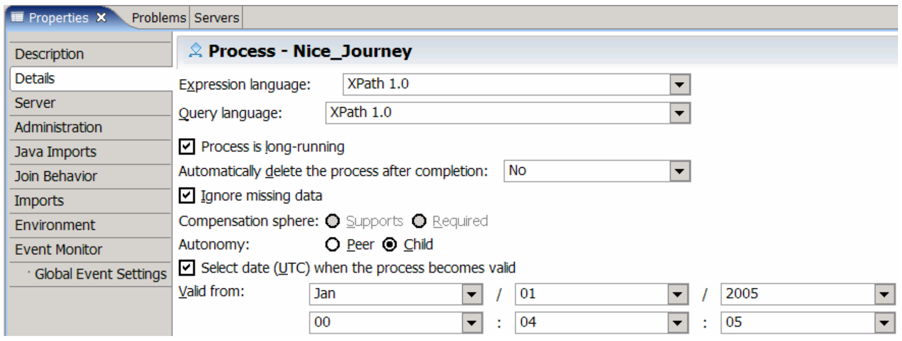
[51] Note that the prefix "wpc" denotes IBM proposed BPEL extensions.

BPEL source code:

```
<bpws:process …
        wpc:executionMode="longRunning" …
        expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116" ...
        wpc:ignoreMissingData="yes" …
        wpc:autonomy="child" …
        suppressJoinFailure="yes" … >
```

Find these properties on the pages Details and Join Behavior of the Properties view:

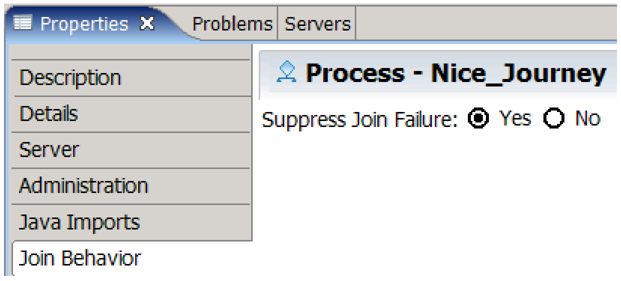
WID business process editor:



Process - Nice_Journey

Expression language: XPath 1.0
Query language: XPath 1.0
[✓] Process is long-running
Automatically delete the process after completion: No
[✓] Ignore missing data
Compensation sphere: ○ Supports ○ Required
Autonomy: ○ Peer [✓ Child]
[✓] Select date (UTC) when the process becomes valid
Valid from: Jan / 01 / 2005
00 : 04 : 05

Figure 33: Mapping process properties (2)



Process - Nice_Journey

Suppress Join Failure: [✓ Yes] No

Figure 34: Mapping process properties (3)

The next sections explain some of the properties in more detail.

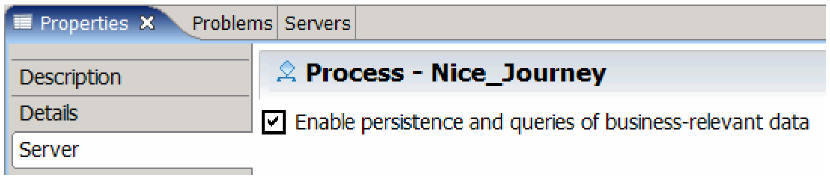
Process property "business relevant"

If the FDL process has property "AUDIT_FILTER_DB" or property "AUDIT_FILTER_MQ", the FDL2BPEL Conversion Tool sets the "businessRelevant" property of the generated BPEL process to "yes". Otherwise, the "businessRelevant" property is set to "no" (unchecked):

BPEL source code:

```
<bpws:process … wpc:businessRelevant="no" … >
```

WID business process editor:



Process - Nice_Journey

☑ Enable persistence and queries of business-relevant data

Figure 35: Mapping process properties (4)

Process autonomy

In WMQWF Buildtime you can select the option **control** for process autonomy on the process settings page Control. It has the following meaning:

> If the process runs as a subprocess of another one, then **terminate**, **suspend**, and **resume** requests for activity and process instances from the parent process will be ignored. This includes terminate requests for activity instances and suspend or terminate requests for process instances.

The FDL2BPEL Conversion Tool maps the process autonomy flag **control** to the corresponding BPEL autonomy flag **peer**. In all other cases, the BPEL autonomy flag is set to **child**.[52]

Process category

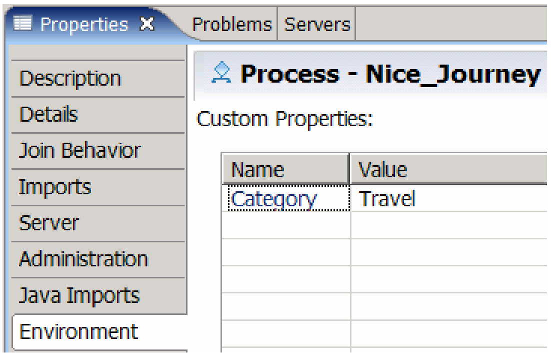
The category that you assigned to your WMQWF model maps to a BPEL extension element "wpc:customProperty".

BPEL source code:

```
<wpc:customProperty name="Category">
   <wpc:value>Travel</wpc:value>
</wpc:customProperty>
```

You find property "Category" on page Environment of the WID Properties view:

WID business process editor:



Process - Nice_Journey

Custom Properties:

Name	Value
Category	Travel

Figure 36: Process Category

[52] See limitation "Process autonomy modes" on page 141.

Partner links

The services that a business process uses are modeled in BPEL as partner links. Each partner link is characterized by a partner link type. The following BPEL code outlines partner links that are generated for a process "Nice_Journey" which has a subprocess "NJ_BookCar". The process "Nice_Journey" has some UPES activities which invoke the servers "FMCSYS_FLIGHT", "FMCSYS_DUMMY01", "FMCSYS_UPES1", and "FMCSYS_UPES2":

BPEL source code:

```
<bpws:partnerLinks>
    <!-- Partner link used to invoke process "Nice_Journey" -->
    <bpws:partnerLink name="Nice_Journey_PL"
                        partnerLinkType="wsdl1:Nice_Journey_PLT"
                        myRole="Nice_Journey_Role"/>
    <!-- Partner link used to invoke process "NJ_BookCar" -->
    <bpws:partnerLink name="NJ_BookCar_PL"
                        partnerLinkType="wsdl1:NJ_BookCar_PLT"
                        partnerRole="NJ_BookCar_Role">
        <wpc:processResolver processTemplateName="NJ_BookCar"/>
    </bpws:partnerLink>
    <!-- Partner link used to invoke server "DUMMY01" -->
    <bpws:partnerLink name="FMCSYS_DUMMY01_PL"
                        partnerLinkType="wsdl1:FMCSYS_DUMMY01_PLT"
                        partnerRole="FMCSYS_DUMMY01_Role"/>
    <!-- Partner link used to invoke server "FLIGHT" -->
    <bpws:partnerLink name="FMCSYS_FLIGHT_PL"
                        partnerLinkType="wsdl1:FMCSYS_FLIGHT_PLT"
                        partnerRole="FMCSYS_FLIGHT_Role"/>
    <!-- Partner link used to invoke server "UPES1" -->
    <bpws:partnerLink name="FMCSYS_UPES1_PL"
                        partnerLinkType="wsdl1:FMCSYS_UPES1_PLT"
                        partnerRole="FMCSYS_UPES1_Role"/>
    <!-- Partner link used to invoke server "UPES2" -->
    <bpws:partnerLink name="FMCSYS_UPES2_PL"
                        partnerLinkType="wsdl1:FMCSYS_UPES2_PLT"
                        partnerRole="FMCSYS_UPES2_Role"/>
</bpws:partnerLinks>
```

The WID BPEL editor displays the partner links as follows:

WID business process editor:

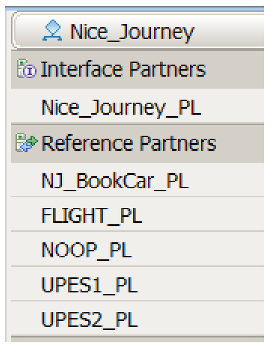


Figure 37: Partner links

Mapping FDL data container to BPEL

BPEL does not have the concept of a "data container" as it is known in WMQWF. Instead, BPEL provides "variables" to hold messages that constitute the state of a business process. Accordingly, the FDL2BPEL Conversion Tool generates BPEL variables as a representation of data containers. That is, the data input and output of the process as well as the data input and output of each contained activity maps onto a corresponding pair of BPEL variables. A variable name consists of the concatenation of the name of the process or activity and a name suffix that indicates the role of an input or output container. The following example shows the generated BPEL variables that are required for a simple process that contains three activities:

BPEL source code:

```
<!-- *** Variables *** -->
<bpws:variables>
    <!-- Data input variable of process "SimpleProcessWDF" -->
    <bpws:variable name="SimpleProcessWDF_IN" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data output variable of process "SimpleProcessWDF" -->
    <bpws:variable name="SimpleProcessWDF_OUT" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data input variable of program activity "Act1" -->
    <bpws:variable name="Act1_IN" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data output variable of program activity "Act1" -->
    <bpws:variable name="Act1_OUT" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data input variable of program activity "Act2" -->
    <bpws:variable name="Act2_IN" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data output variable of program activity "Act2" -->
    <bpws:variable name="Act2_OUT" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data input variable of program activity "Act3" -->
    <bpws:variable name="Act3_IN" type="mqwf:TravelRequest_DS_MT"/>
    <!-- Data output variable of program activity "Act3" -->
    <bpws:variable name="Act3_OUT" type="mqwf:TravelRequest_DS_MT"/>
</bpws:variables>
```

WID business process editor:

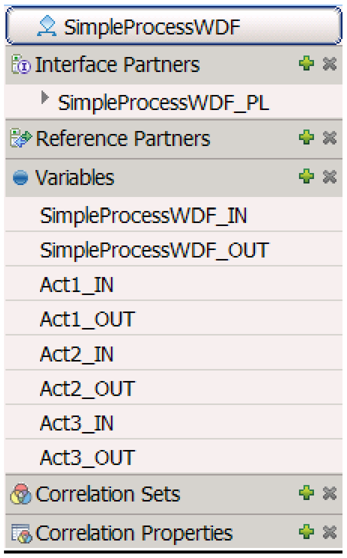


Figure 38: Mapping FDL data containers to BPEL variables

Queriable global data container

In WMQWF you can specify a special data container, called "global data container", which you can refer to in queries (in addition to references to the predefined properties of the entities exposed by the query API). In BPEL, you can assign "query properties" to a BPEL variable in order to have a similar concept. When the FDL2BPEL Conversion Tool encounters the definition of a global container in the FDL, it maps it onto a BPEL variable with the name "GLOBAL_CONTAINER".

Here is an example that shows how an FDL global container with a related data structure "customer Info" is mapped onto a corresponding BPEL variable:

FDL specification of global data container:

```
STRUCTURE 'customer Info'
  'Family name': STRING;
  'Company': STRING;
  'City': STRING;
  'Street': STRING;
  'Phone': STRING;
END 'customer Info'
PROCESS 'MyProcess' ( 'Customer', 'Customer' )
  GLOBAL_CONTAINER RELATED_STRUCTURE 'customer Info'
    TABLE_NAME "GC_MY_TABLE"
    INDEX INDEX_NAME "GC_MY_INDEX"
      RELATED_STRUCTURE_MEMBER 'Family name'
    …
```

BPEL source code:

```
<!-- "Global Container" variable -->
<bpws:variable name="GLOBAL_CONTAINER" type="mqwf:CustomerInfo_GC_MT">
        <wpc:queryProperties>
                <wpc:queryProperty name="Familyname" type="xsd:string">
                        <wpc:query><![CDATA[_STRUCT/Familyname]]></wpc:query>
                </wpc:queryProperty>
                <wpc:queryProperty name="Company" type="xsd:string">
                        <wpc:query><![CDATA[_STRUCT/Company]]></wpc:query>
                </wpc:queryProperty>
                <wpc:queryProperty name="City" type="xsd:string">
                        <wpc:query><![CDATA[_STRUCT/City]]></wpc:query>
                </wpc:queryProperty>
                <wpc:queryProperty name="Street" type="xsd:string">
                        <wpc:query><![CDATA[_STRUCT/Street]]></wpc:query>
                </wpc:queryProperty>
                <wpc:queryProperty name="Phone" type="xsd:string">
                        <wpc:query><![CDATA[_STRUCT/Phone]]></wpc:query>
                </wpc:queryProperty>
        </wpc:queryProperties>
</bpws:variable>
```

WID business object editor:

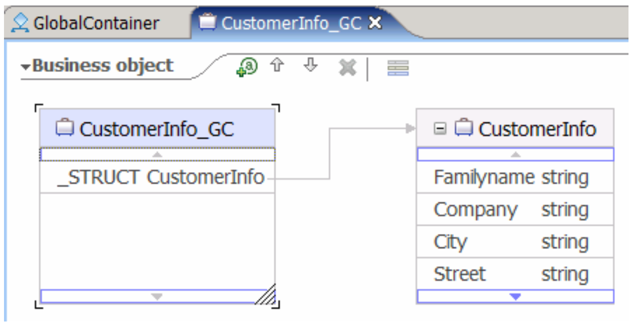
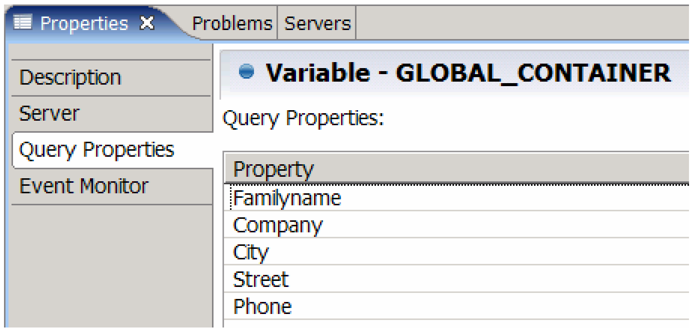


Figure 39: Mapping Global Container to business object

WID Properties view of BPEL variable "GLOBAL_CONTAINER":



Variable - GLOBAL_CONTAINER

Query Properties:

Property
Familyname
Company
City
Street
Phone

Figure 40: Mapping FDL data structure to BPEL query properties

Mapping the FDL workflow to BPEL

The following BPEL activity framework outlines the general structure that is used by the FDL2BPEL Conversion Tool to represent every FDL workflow[53]:

BPEL source code:

```
<sequence...>
      <!-- Receive the input data of process "processname" -->
      <receive name="Receive_the_process_input">
            <wpc:output>
                  <wpc:parameter variable="processname_IN" name="input1"/>
            </wpc:output>
      </receive>
      <!-- Implementation of process "processname" -->
      <flow name="Execute_activities_contained_in_process"
              wpc:displayName="Execute activities contained in process">
      ...
      </flow>
      <!-- Send the output data of process "processname" -->
      <reply name="Return_the_process_output ">
            <wpc:input>
                  <wpc:parameter variable="processname_OUT" name="output1"/>
            </wpc:input>
      </reply>
</sequence>
```

[53] See also "Optimizing BPEL process models" on page 98

BPEL business process editor:

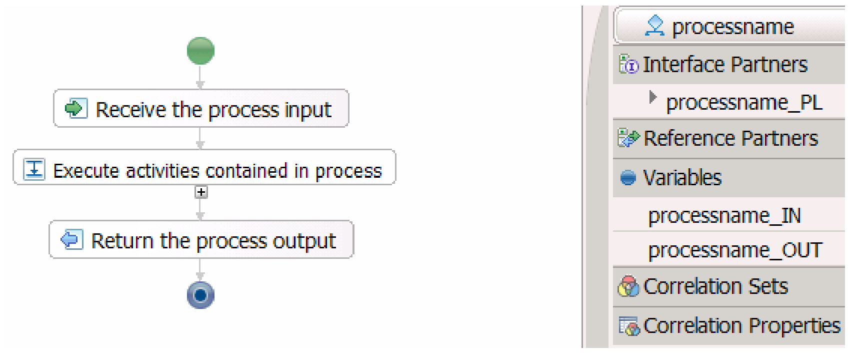


Figure 41: Mapping the workflow structure (1)

The process input data is "received" by a <receive…> activity in variable "*processname*_IN". The process result data is provided in the variable "*processname*_OUT" and returned to the service requester by a <reply> activity. The following <flow…> activity ("Execute activities contained in process") contains the activity network of the workflow. A <sequence...> activity ensures the appropriate execution order of this procedure.

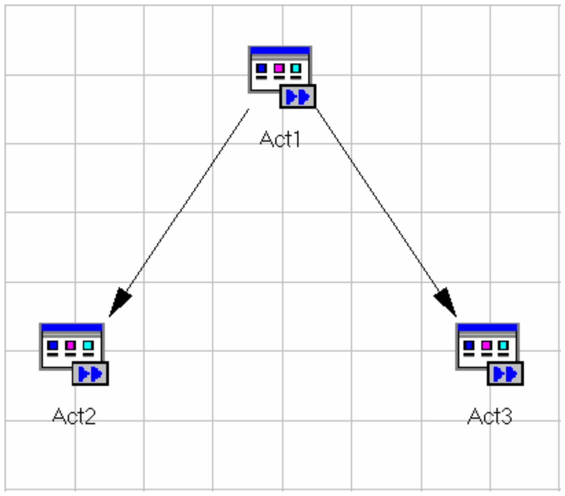


Figure 42: Mapping the workflow structure (2)

The following BPEL code outlines the internals of the <flow…> that represents the above FDL activity network consisting of three FDL activities (example without data flow):

```
<!-- Implementation of process "SimpleProcess" -->
<flow >
      <!—
            Links that implement the internal control flow
            of process "SimpleProcess"
      -->
      <links>
            <link name="Act1-to-Act2"/>
            <link name="Act1-to-Act3"/>
      </links>
      …
      <!-- Activity "Act1" -->
      <invoke name="Act1">
            <sources>
                  <source linkName="Act1-to-Act2"/>
                  <source linkName="Act1-to-Act3"/>
            </sources>
            …
      </invoke>
      …
      <!-- Activity "Act2" -->
      <invoke name="Act2">
            <targets>
                  <target linkName="Act1-to-Act2"/>
            </targets>
            …
      </invoke>
      …
      <!-- Activity "Act3" -->
      <invoke name="Act3">
            <targets>
                  <target linkName="Act1-to-Act3"/>
            </targets>
            …
      </invoke>
                        …
</flow>
```

Note that, in contrast to the FDL notation, BPEL <link…> elements have no attributes that specify the source and target activities. BPEL <link…> elements always carry names that are referenced by <source…> and <target…> child elements of the activities that are involved in the control flow.

The figure below shows the graphical representation of the above flow in the WID BPEL editor when all initially collapsed activities are expanded:

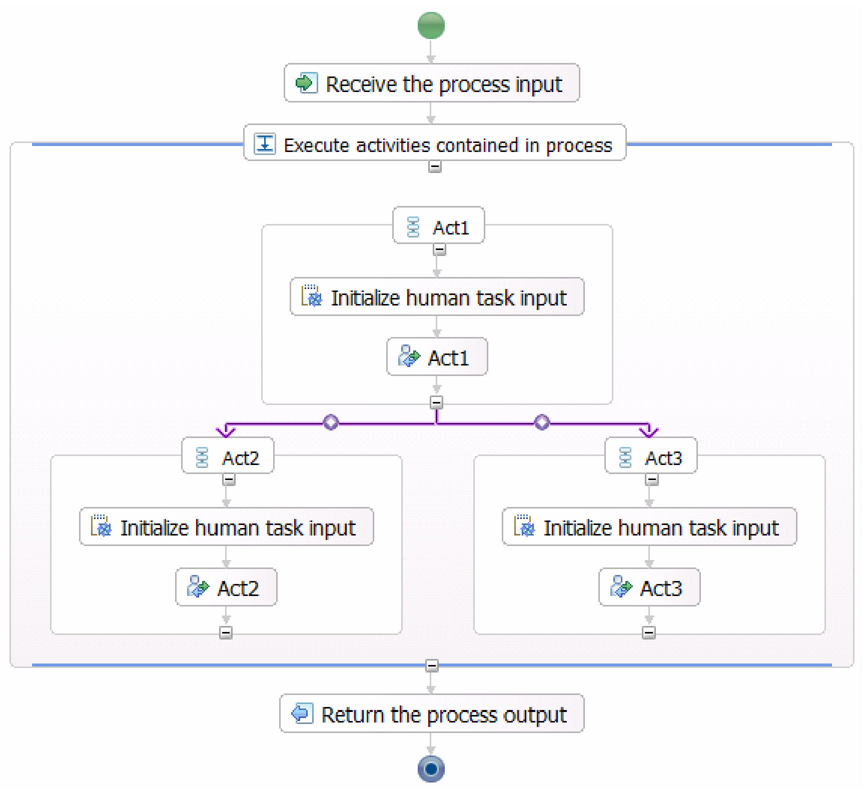

Receive the process input
Execute activities contained in process
Act1
Initialize human task input
Act1
Act2
Initialize human task input
Act2
Act3
Initialize human task input
Act3
Return the process output

Figure 43: Mapping the workflow structure (3)

The FDL program activities "Act1", "Act2", and "Act3" are mapped to corresponding BPEL <invoke...> activities that are encapsulated in BPEL <sequence...> structured activities. In the above sample process without data flow, the <sequence...> activities and the "Initialize human task input" snippets are not actually necessary. However, in more complex models the <sequence...> construct acts as a necessary wrapper that ensures the proper order of data flow and execution.

Mapping WMQWF activities to BPEL

The FDL2BPEL Conversion Tool maps activities to BPEL <invoke...> activities. The <invoke...> construct allows the business process to invoke a request-response (synchronous) or a one-way (asynchronous) operation on a portType offered by a partner.

Mapping FDL PROGRAM_ACTIVITY to BPEL

BPEL and the extensions defined by IBM do not allow an exact representation of an FDL program activity (see migration limitation "FDL ProgramActivity" on page 142). For instance, FDL lets you describe the assignment of staff and a program in a single program activity. The current BPEL specification has no equivalent definition for this. That is why the FDL2BPEL Conversion Tool applies a set of activity classification rules that control the mapping of FDL program activities in categories "empty activity", "service invocation activity", and "staff activity".

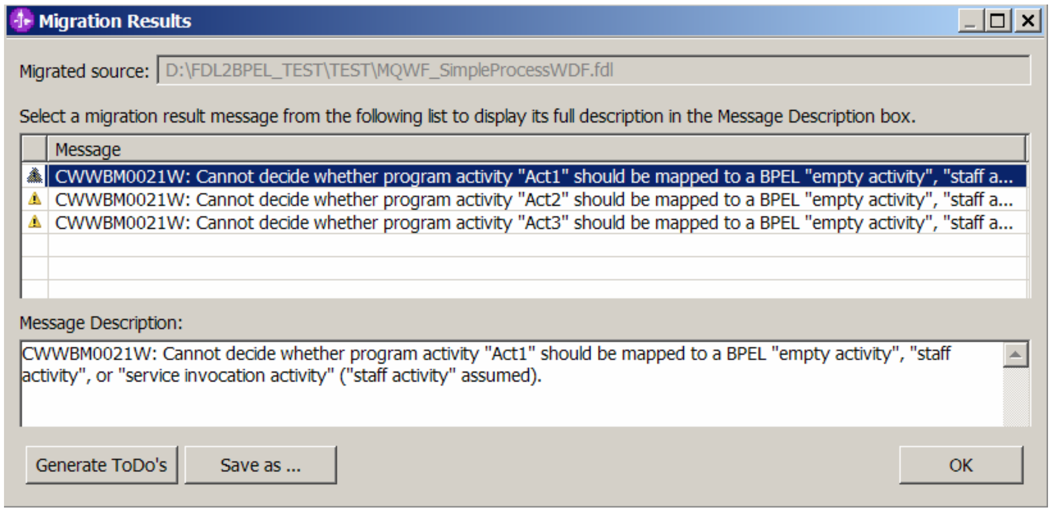
Activity classification rules

The following rules apply (in the order listed) for mapping FDL program activities to BPEL activities:

Table 5: Activity classification rules

	Activity type[54]	Mapping rule
1.	**"Empty activity"**	A program activity maps to an "empty activity" if 1. the activity implementation is a program with the name "FMCINTERNALNOOP", 2. the execution mode is asynchronous[55] (see the properties notebook page "Execution"), and 3. the input and output data structures are the same.
2.	**"Service invocation activity"**	A program activity maps to a "service invocation activity", if it is associated to a user-defined program execution server (UPES). That is, you specified a program execution server on page "Execution" of the program activity properties notebook.
3.	**"Staff activity"**	A program activity maps to a "staff activity" if 1. its start type is "manual" (see the properties notebook page "Start"), and 2. the property "Program activities can be checked out" indicates that it can be checked out from the runtime database (see the properties notebook page "Control").

If none of the above rules match, you receive a warning and the activity will be assumed to be a "staff activity":


Migration Results
Migrated source: D:\FDL2BPEL_TEST\TEST\MQWF_SimpleProcessWDF.fdl
Select a migration result message from the following list to display its full description in the Message Description box.
Message
Message Description:
Generate ToDo's
Save as ...
OK

```
CWWBM0021W: Cannot decide whether program activity "Act1" should be mapped to a BPEL "empty activity", "staff a...
CWWBM0021W: Cannot decide whether program activity "Act2" should be mapped to a BPEL "empty activity", "staff a...
CWWBM0021W: Cannot decide whether program activity "Act3" should be mapped to a BPEL "empty activity", "staff a...
```

CWWBM0021W: Cannot decide whether program activity "Act1" should be mapped to a BPEL "empty activity", "staff activity", or "service invocation activity" ("staff activity" assumed).

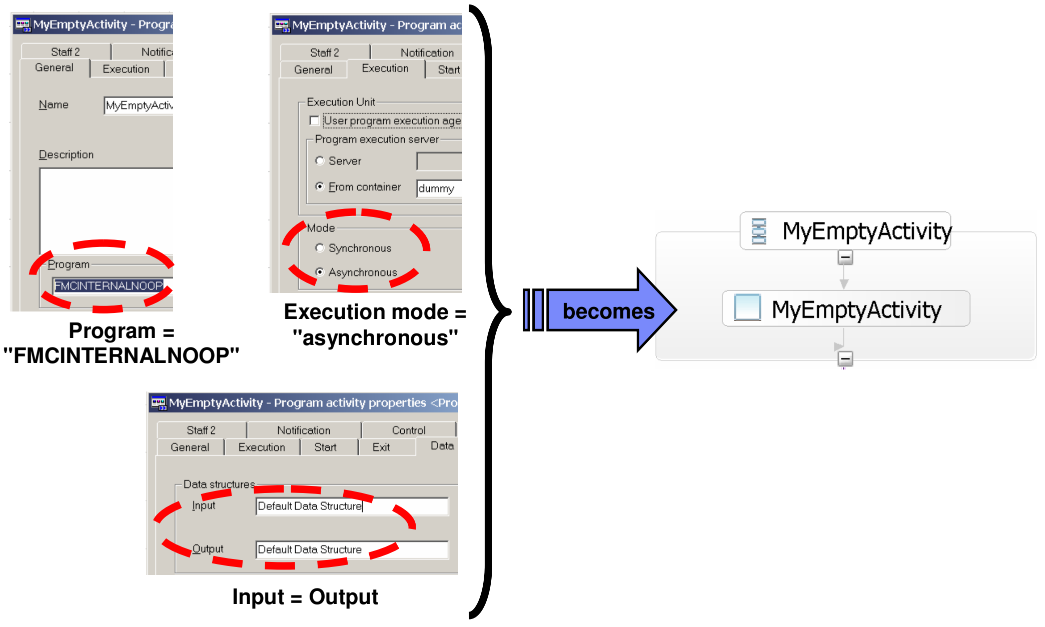
Figure 44: Assumed "staff activity"

[54] Note that the activity types listed here are FDL2BPEL internal notions and do not necessarily match the terms that you encounter when modeling a business process using the WebSphere Integration Developer.

[55] Because you cannot specify asynchronous mode without defining an execution server, you can do so by entering a "dummy" server name.

Empty activity

Here is an example of the FDL activity settings and the mapping to BPEL of an "empty activity":



Name MyEmptyActiv

Program FMCINTERNALNOOP

Mode: Asynchronous

Input Default Data Structure

Output Default Data Structure

Program = "FMCINTERNALNOOP"

Execution mode = "asynchronous"

Input = Output

Figure 45: Activity classification rule: "empty activity"

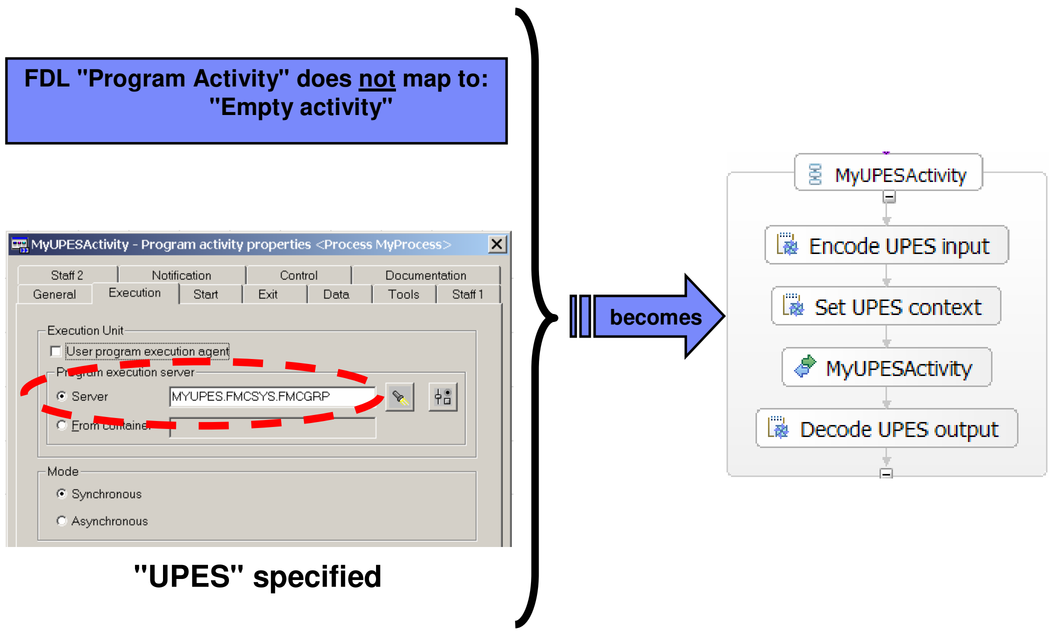
<u>BPEL source code:</u>

```
<empty name="EmptyActivity"/>
```

Service invocation activity (UPES activity)

The following example shows the BPEL translation of an FDL PROGRAM_ACTIVITY that was classified as a "service invocation activity". As the service is related to a user-defined program execution server (UPES), it is also called a "UPES activity":[56]

[56] See also chapter "WMQWF UPES migration" on page 97.



Execution Unit
User program execution agent [ ]
Program execution server
Server [✓] MYUPES.FMCSYS.FMCGRP
From container [ ]

Mode
Synchronous [✓]
Asynchronous [ ]

Figure 46: Activity classification rule: "service invocation activity"

BPEL source code:

```
<bpws:invoke name="MyUPESActivity"
             operation="MyUPESActivity_UPES"
             portType="wsdl1:FMCSYS_MYUPES_PT"
             partnerLink="FMCSYS_MYUPES_PL"
             wpc:continueOnError="no"
             wpc:displayName="MyUPESActivity"
             wpc:businessRelevant="no">
    <wpc:input>
        <wpc:parameter variable="MyUPESActivity_UPES_IN" name="input1"/>
    </wpc:input>
    <wpc:output>
        <wpc:parameter variable="MyUPESActivity_UPES_OUT" name="output1"/>
    </wpc:output>
</bpws:invoke>
```

The input/output parameter elements refer to BPEL variables that contain the data that is passed to and from the BPEL *invoke* activity. As already mentioned in section "Mapping the data exchanged with a UPES" on page 41, the input and output data for a UPES invocation requires another message structure that consists of "container" and "context" data. This means that the "standard" message must be mapped to a UPES compliant structure before you can invoke the "UPES Web service". Correspondingly, the output message received from a UPES must be mapped back onto the "standard" message structure. In order to perform this conversion task, each "service invocation activity" is assigned another pair of input and output BPEL variables. For example, an activity "MyUPESActivity" is assigned two BPEL variables, "MyUPESActivity_IN" and "MyUPESActivity_OUT", which are used for the "standard" message structure, and another two BPEL variables, "MyUPESActivity_UPES_IN" and "MyUPESActivity_UPES_OUT", which are used for the "UPES" message structure.

As explained in the section "Mapping FDL data flow to BPEL" on page 77, a snippet ("Encode UPES input") is used to map a "standard" message with input business data from the variable "MyUPESActivity_IN" to the variable "MyUPESActivity_UPES_IN" that holds the "UPES" input message, which consists of business data and context information (see the following figure).

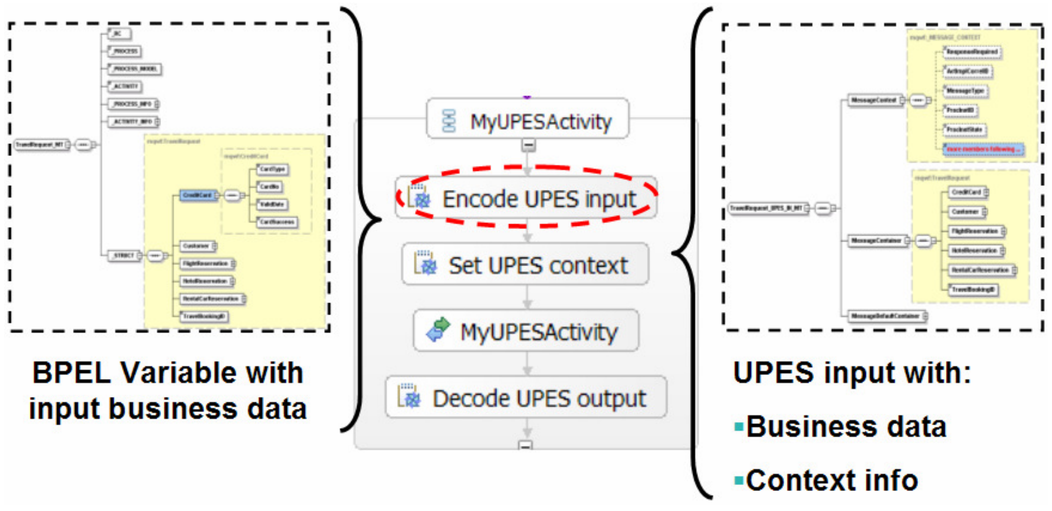


Figure 47: Encoding the UPES input

Another snippet ("Decode UPES output") is required to perform the mapping from the "UPES" output message contained in the BPEL variable "MyUPESActivity_UPES_OUT" to the BPEL variable "MyUPESActivity_OUT":

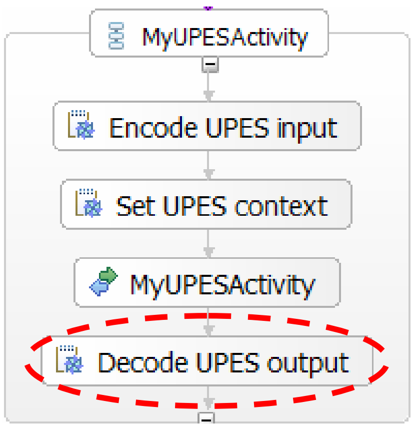


Figure 48: Decoding the UPES output

Note that the graphical representation may also show a snippet "Encode UPES output defaults" (not shown here), which maps the data default values of the BPEL variable "MyUPESActivity_OUT" to the variable "MyUPESActivity_UPES_IN". This snippet is required, if there are any initial values for the activity output container which should be part of the *ProgramOutputDataDefaults* XML element of the UPES message.

The snippet "Set UPES context" adds context data to the UPES message:

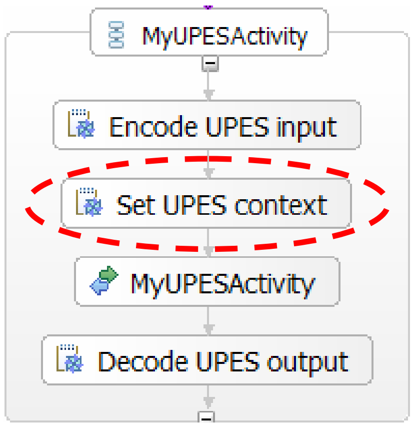


Figure 49: Setting the UPES context

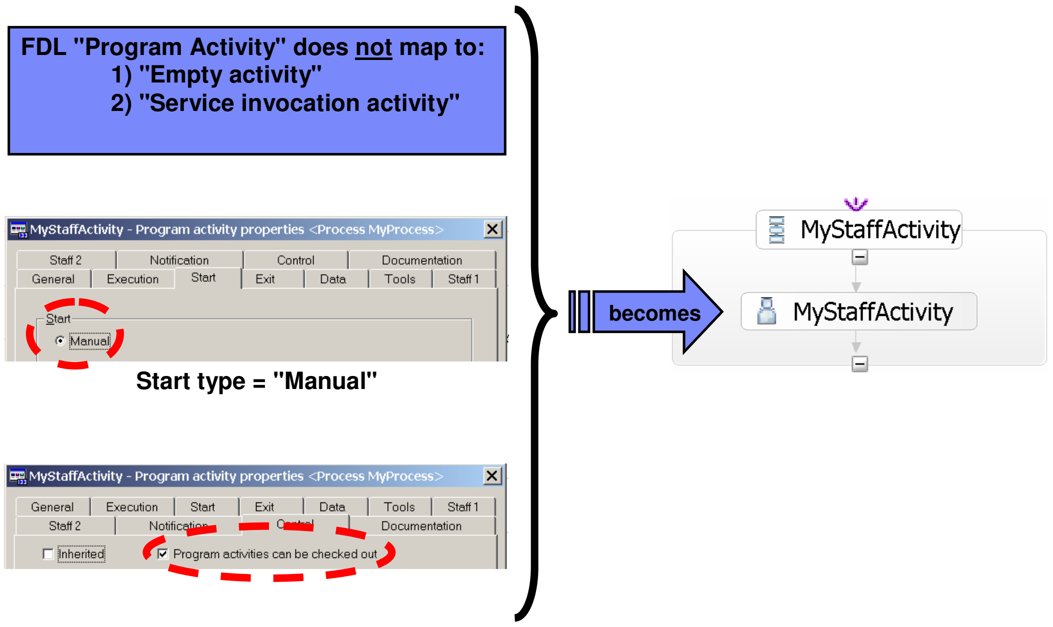
The snippet "Set UPES context" initializes the following elements of the XML message that the UPES will receive:

Table 6: UPES context data

_PROCESS	The process instance name
ResponseRequired	'yes' for synchronous execution mode 'no' for asynchronous execution mode
ActImplCorrelID	The correlation identifier
Starter	The starter of the process instance
ProcTemplID	The process template identifier
ProgramName	The program name

For more information about using snippets to map data flow, refer to the section "Mapping FDL data flow to BPEL" on page 77.

Staff activity


FDL "Program Activity" does not map to:
1) "Empty activity"
2) "Service invocation activity"

MyStaffActivity - Program activity properties <Process MyProcess>
Start: (●) Manual

Start type = "Manual"

MyStaffActivity - Program activity properties <Process MyProcess>
☐ Inherited
☑ Program activities can be checked out

Figure 50: Classification rule "staff activity"

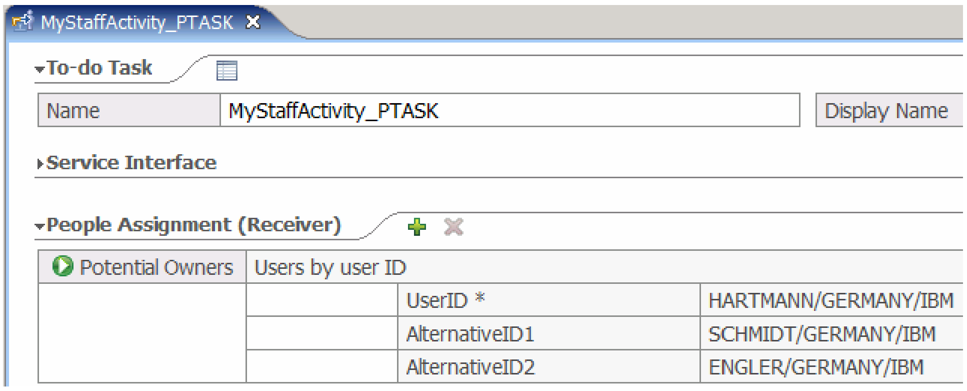
BPEL source code:

```
<!-- Invoke program activity "MyStaffActivity" -->
<bpws:invoke name="MyStaffActivity"
                operation="MyStaffActivity"
                portType="wsdl1:dummy_PT"
                partnerLink="null"
                wpc:continueOnError="no"
                wpc:displayName="MyStaffActivity"
                wpc:businessRelevant="no">
        <!-- Staff assignment (to-do task) for activity "MyStaffActivity" -->
        <wpc:task name="staff:MyStaffActivity_PTASK"/>
        <wpc:input>
                <wpc:parameter variable="MyStaffActivity_IN" name="input1"/>
        </wpc:input>
        <wpc:output>
                <wpc:parameter variable="MyStaffActivity_OUT" name="output1"/>
        </wpc:output>
</bpws:invoke>
```

A "staff activity" is rendered as an extension of the existing BPEL <invoke...> activity by adding the child element <wpc:task…>. The "task" element's name attribute refers to a TEL file that contains the description of a "to-do task". In particular, the task TEL definition consists of "people assignment criteria" that represent the rules for assigning work items to people at runtime (see "Mapping WMQWF staff assignment criteria to TEL people assignment criteria" on page 85):

TEL source code:

```
<tel:staffSettings>
        <tel:potentialOwner>
                <tel:verb>
                        <!-- FDL staff query "People" -->
                        <tel:name>Users by user ID</tel:name>
                        <tel:parameter id="UserID">HARTMANN/GERMANY/IBM</tel:parameter>
                        <tel:parameter id="AlternativeID1">SCHMIDT/GERMANY/IBM</tel:parameter>
                        <tel:parameter id="AlternativeID2">ENGLER/GERMANY/IBM</tel:parameter>
                </tel:verb>
        </tel:potentialOwner>
</tel:staffSettings>
```



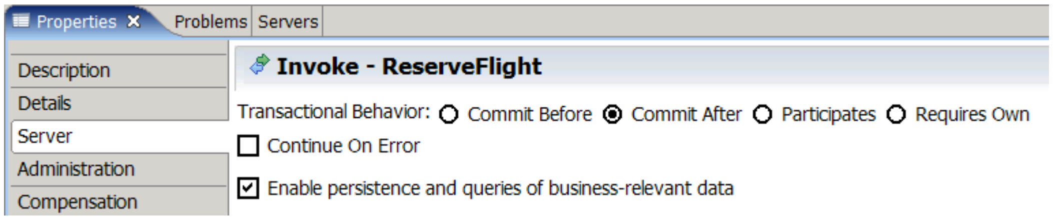
Name	MyStaffActivity_PTASK		Display Name

Potential Owners	Users by user ID	
	UserID *	HARTMANN/GERMANY/IBM
	AlternativeID1	SCHMIDT/GERMANY/IBM
	AlternativeID2	ENGLER/GERMANY/IBM

Figure 51: WID human task editor

Activity property "business relevant"

The FDL2BPEL Conversion Tool sets the "business relevant" property of the generated BPEL "invoce" activity to "yes", if the FDL activity has property "AUDIT_FILTER_DB" or property "AUDIT_FILTER_MQ". Otherwise, the "business relevant" property is set to "no".



Invoke - ReserveFlight

Transactional Behavior: ○ Commit Before [✓ Commit After] ○ Participates ○ Requires Own

☐ Continue On Error

☑ Enable persistence and queries of business-relevant data

Figure 52: Mapping activity property "business relevant"

Mapping FDL BLOCK to BPEL

An FDL block activity allows process decomposition by encapsulating underlying activities, and simplifying the visual process model by replacing the underlying sequence of logic with a single block activity. This maps nicely to a BPEL "Parallel Activities" node. The figure below shows the mapping of an FDL block that contains a simple flow of three activities:

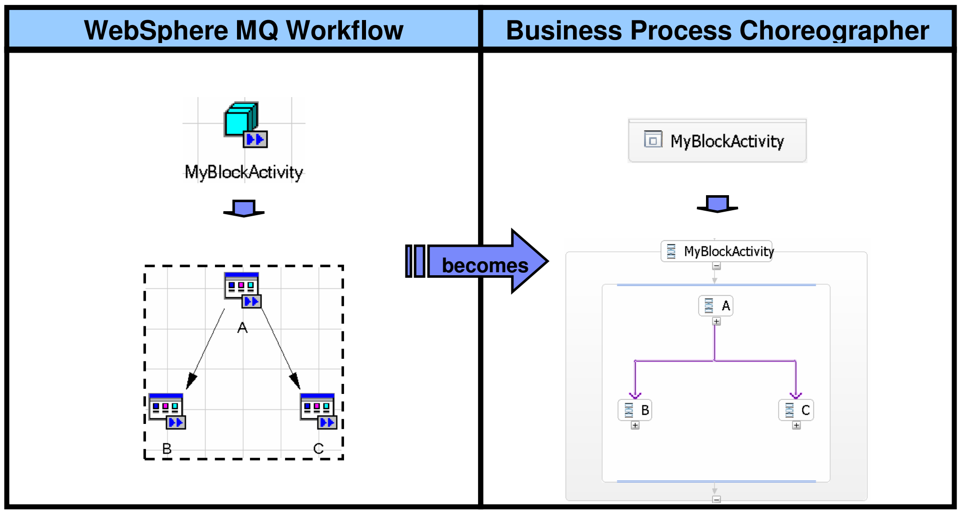


Figure 53: Mapping an FDL block to BPEL

An FDL block allows repeated activities to be modeled so that block enclosed activities are executed at least once, and will loop until an exit condition is true. The migration tool translates an exit condition into a BPEL <while…> activity (see "Exit conditions" on page 73 for further details).

Mapping FDL PROCESS_ACTIVITY to BPEL

Static subprocess invocation

FDL allows the nesting of processes by embedding subprocess calls. A subprocess is actually a normal FDL process that is invoked within another process. Again this invocation mechanism maps nicely into the BPEL service invocation framework, where each BPEL process is just another service, and hence can also be nested. Thereby an FDL subprocess invocation is translated into an <invoke...> activity that is assigned to a partner link. This is illustrated in the figure below:

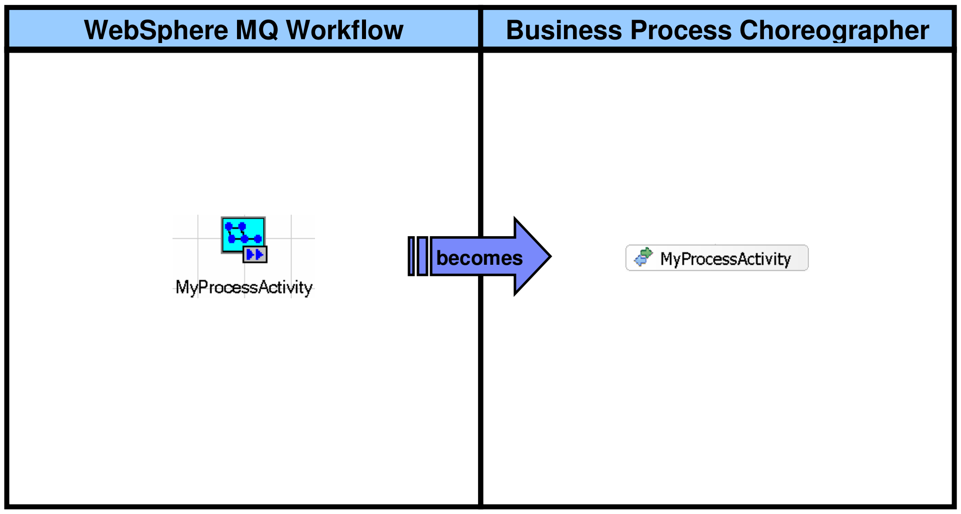
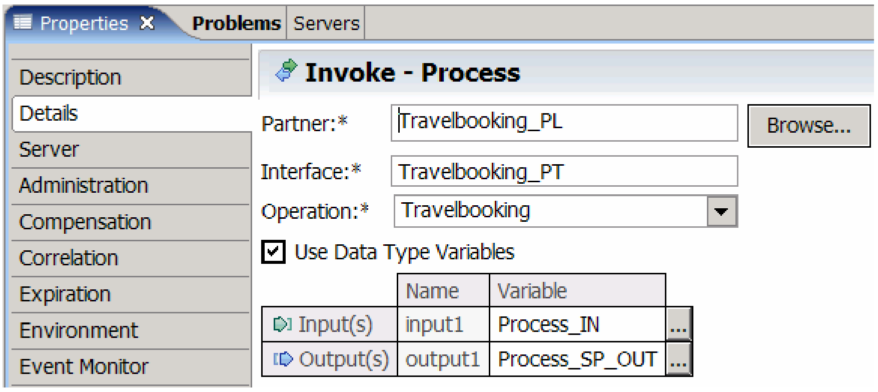


Figure 54: Mapping an FDL subprocess call to BPEL

The following figure shows the invoke implementation properties sheet:



Partner:* Travelbooking_PL

Interface:* Travelbooking_PT

Operation:* Travelbooking

[✓] Use Data Type Variables

	Name	Variable
Input(s)	input1	Process_IN
Output(s)	output1	Process_SP_OUT

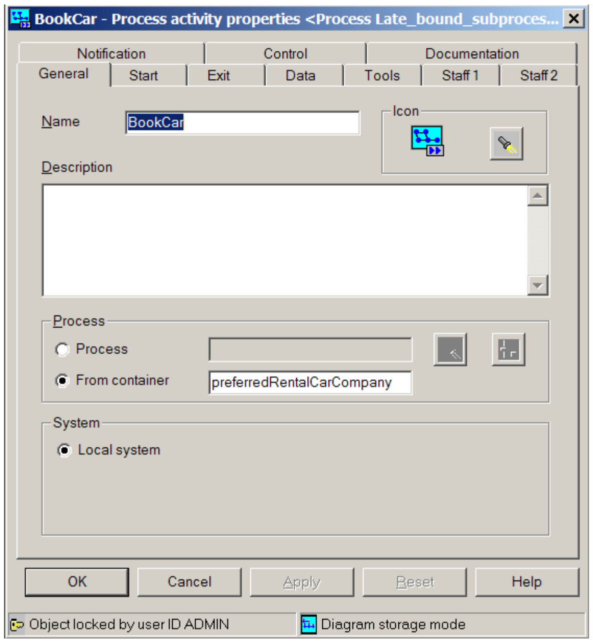
Figure 55: Subprocess invoke properties

Late binding of subprocess names

FDL allows you to specify the name of the process template to use for a subprocess in a container data member. Rather than having a fixed value, this container value can be modified on behalf of the process execution. For example:

In BPEL, the corresponding mechanism is to use an assign activity that modifies the partner link data. Thus, the FDL setting "**Process from container**" is mapped onto an additional assign activity with the following mapping specification.

Assume that in an FDL process activity "BookCar" that allows for a late-bound process, process names "BookCompanyA", "BookCompanyB", "BookCompanyC", etc. exist with the process name taken from the input data field "preferredRentalCarCompany" (see figure below):



Name BookCar

Process

( ) Process

(●) From container preferredRentalCarCompany

System

(●) Local system

Object locked by user ID ADMIN

Diagram storage mode

Figure 56: Process from container

The corresponding BPEL diagram shows that the <assign> activity that binds the subprocess name can be inserted into a <sequence...> structured activity just before the "BookCar" <invoke...> activity:

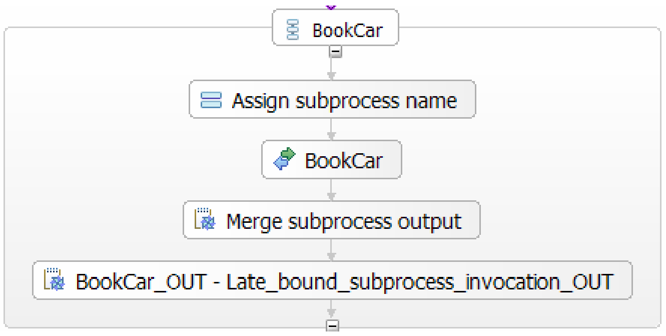


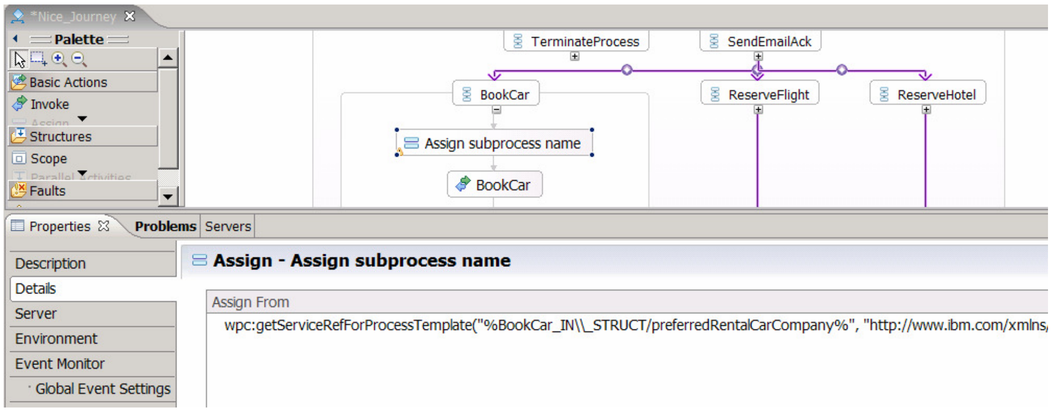
Figure 57: Late-bound subprocess invocation

BPEL source code:

```
<bpws:partnerLink name="Late_bound_subprocess_invocation_BookCar_PL"
                  partnerLinkType="wsdl1:Late_bound_subprocess_invocation_BookCar_PLT"
                  partnerRole="Late_bound_subprocess_invocation_BookCar_Role"/>
  ...

<!-- Assign late-bound subprocess name for process activity "BookCar" -->
<bpws:assign name="gen0024" wpc:displayName="Assign subprocess name">
    <bpws:copy>
        <bpws:from>
            <bpws:expression expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116">
                wpc:getServiceRefForProcessTemplate(
                                                    "%BookCar_IN\\_STRUCT/preferredRentalCarCompany%",
                                                    "http://www.ibm.com/xmlns/prod/websphere/mqwf/wsdl/",
                                                    "BookCar_PL")
            </bpws:expression>
        </bpws:from>
        <bpws:to partnerLink="BookCar_PL"/>
</bpws:assign>
…
<!-- Invoke process activity "BookCar" -->
<bpws:invoke  name="BookCar"
              operation="BookCar"
              portType="wsdl1:BookCar_PT"
              partnerLink="BookCar_PL"
              wpc:continueOnError="no"
              wpc:displayName="BookCar"
              wpc:businessRelevant="no">
    <wpc:input>
        <wpc:parameter variable="BookCar_IN" name="input1"/>
    </wpc:input>
    <wpc:output>
        <wpc:parameter variable="BookCar_SP_OUT" name="output1"/>
    </wpc:output>
</bpws:invoke>
…
```

WID BPEL editor with properties view of assign activity:



Assign - Assign subprocess name

Assign From

```
wpc:getServiceRefForProcessTemplate("%BookCar_IN\\_STRUCT/preferredRentalCarCompany%", "http://www.ibm.com/xmlns,
```

Figure 58: Assigning the subprocess name

Mapping WMQWF control flow to BPEL

In "Mapping the FDL workflow to BPEL" on page 57, you already learned about the usage of <link> elements which are used to convert FDL control connectors. <link> elements represent the static structure of synchronization dependencies between activities. This section explains how the dynamic synchronization dependencies specified by conditions and activity events like expiration and notification are converted to BPEL by the FDL2BPEL Conversion Tool.

Transition conditions

A transition condition is a logical expression that describes when control is transferred from the source activity to the target activity of a control connector. An FDL transition condition is mapped to an equivalent BPEL element <transitionCondition> that is added as a child of the <source> element of the activity. The following example shows the FDL specification of a transition condition and the corresponding translation to BPEL:

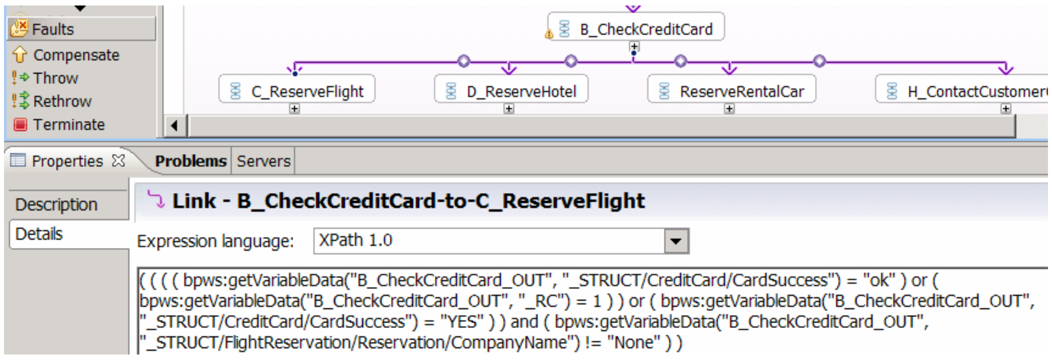
FDL source code:

```
CONTROL
  FROM 'B_CheckCreditCard' TO 'C_ReserveFlight'
  WHEN "(CreditCard.CardSuccess= ""ok""
          OR _RC= 1
          OR CreditCard.CardSuccess= ""YES"" )
          AND FlightReservation.Reservation.CompanyName<> ""None"" "
```

<u>BPEL source code:</u>

```
<source linkName="B_CheckCreditCard-to-C_ReserveFlight">
    <transitionCondition
              expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116">
     (
           (
             (
                 ( getVariableData("Travelbooking_B_CheckCreditCard_OUT",
                                    "_STRUCT/CreditCard/CardSuccess")
                                    = "ok" )
              or ( getVariableData("Travelbooking_B_CheckCreditCard_OUT",
                                  "_RC")
                                  = 1 )
             )
             or ( getVariableData("Travelbooking_B_CheckCreditCard_OUT",
                                 "_STRUCT/CreditCard/CardSuccess")
                                 = "YES" )
           )
        and ( getVariableData("Travelbooking_B_CheckCreditCard_OUT",
                             "_STRUCT/FlightReservation/Reservation/CompanyName")
                             != "None" )
        )
    </transitionCondition>
</source>
```

<u>WID BPEL editor with properties view of link connector:</u>



Link - B_CheckCreditCard-to-C_ReserveFlight

Expression language: XPath 1.0

```
( ( ( ( bpws:getVariableData("B_CheckCreditCard_OUT", "_STRUCT/CreditCard/CardSuccess") = "ok" ) or (
bpws:getVariableData("B_CheckCreditCard_OUT", "_RC") = 1 ) ) or ( bpws:getVariableData("B_CheckCreditCard_OUT",
"_STRUCT/CreditCard/CardSuccess") = "YES" ) ) and ( bpws:getVariableData("B_CheckCreditCard_OUT",
"_STRUCT/FlightReservation/Reservation/CompanyName") != "None" ) )
```

Figure 59: Mapping a transition condition expression to BPEL

Start conditions

WMQWF applies the concept of a *start condition* to specify requirements about concurrent control flow paths that reach an activity. A start condition may have the values:

- At least one incoming connector true (logical **OR**)
- All incoming connectors true (logical **AND**)

At runtime, the MQ Workflow flow engine evaluates the transition conditions for the activity's incoming connectors. If the start condition is true:

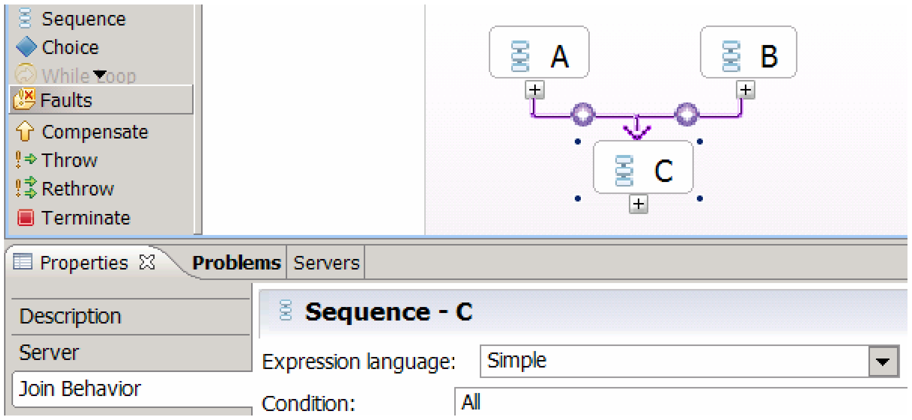
- If it is a manual activity, it is put on to the worklist of the designated staff.
- If it is an automatic activity, it is started.

The corresponding construct in BPEL is a join condition. BPEL actually allows arbitrarily complex Boolean expressions for the join condition. In converting from FDL to BPEL, it is sufficient to use the conditions "any" and "all". The following BPEL code fragment shows how a start condition with the value "All incoming connectors true" is translated into the "built-in" condition value "all":

BPEL source code:

```
<!-- Activity "C" -->
<bpws:sequence name="gen0006" wpc:displayName="C">
    <bpws:targets>
        <bpws:joinCondition expressionLanguage="http://www.ibm.com/xmlns/...">
            <wpc:all/>
        </bpws:joinCondition>
        <bpws:target linkName="A-to-C"/>
        <bpws:target linkName="B-to-C"/>
    </bpws:targets>
    …
```

WID BPEL editor with details view of start condition:



Sequence - C
Expression language: Simple
Condition: All

Figure 60: Mapping a start condition expression to BPEL

Exit conditions

The exit condition for a WMQWF activity is a logical expression that must evaluate to "true", for the activity to end. The evaluation result "false" causes the activity to be repeated. The exit condition is therefore an implicit representation of a control flow loop, which would otherwise be forbidden, if you modeled it explicitly in the workflow model.

BPEL does not have the concept of an exit condition. The only way to create a semantically equivalent representation is to use a structured activity of type "*While loop*", which supports looping on a specified iterative activity. The iterative activity is performed while the given Boolean condition evaluates to true. Here is a BPEL syntax outline of a *While Loop* activity:

BPEL source code:

```
<while>
    <condition
        expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116">
                … XPath expression comes here …
    </condition>
        activity
</while>
```

Note that the FDL2BPEL mapping logic has to consider the following semantic differences between FDL *exit* conditions and BPEL *while* conditions:

Table 7: Mapping an exit condition expression to BPEL

FDL *exit* condition	BPEL *while* condition
…must evaluate to "**true**", in order to exit the loop iteration.	…must evaluate to "**false**", in order to exit the loop iteration.
… is evaluated **after** the activity is executed.	… is evaluated **before** the activity is executed.

The FDL2BPEL Conversion Tool compensates for the differences, as follows:

1. Inverts (negates) the truth value that results from an FDL activity exit condition.
2. Uses an auxiliary BPEL "activity-status" variable to enforce at least one iteration cycle of the *while* loop.

For example, given an FDL exit condition *OrderStatus<>"postponed":*

FDL source code:

```
EXIT WHEN "OrderData.OrderStatus<>""postponed"""
```

The pseudocode for the equivalent *while* condition is:

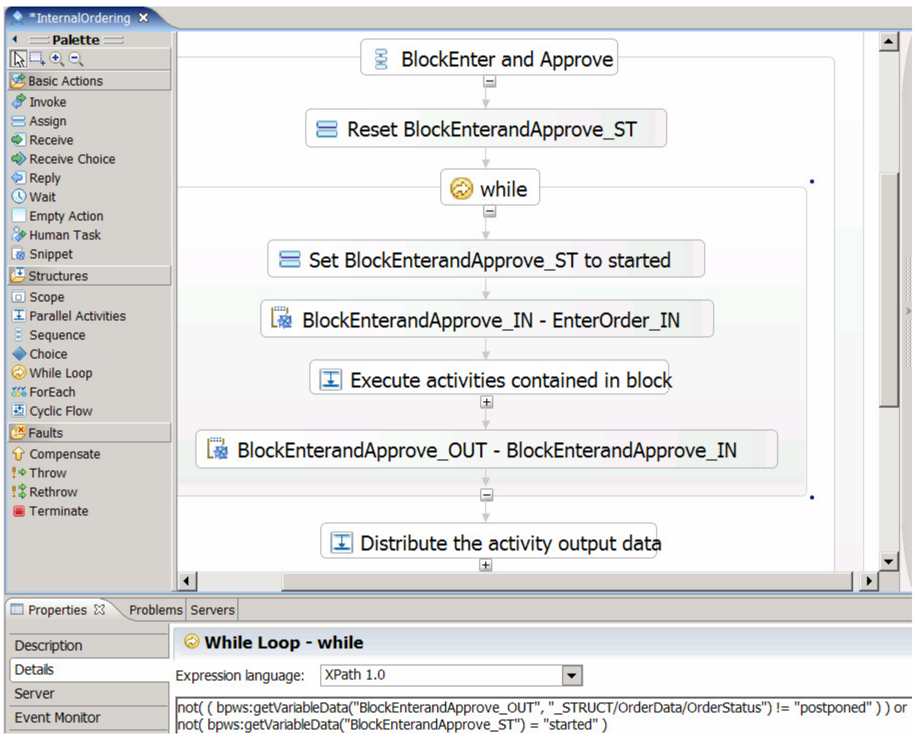
```
NOT(Activity-Status = "started") OR NOT(OrderStatus<>"postponed")
```

If you initialize the BPEL *Activity-Status* variable with a value other than "started" (such as "NOT SET") the iterative activity inside the *while* construct will be performed at least once. The FDL2BPEL Conversion Tool generates "assign" activities that initialize and reset the activity status variables. The following BPEL code provides an example of a generated BPEL *while* condition using an XPath expression:

BPEL source code:

```
<bpws:while name="gen0011" wpc:displayName="while">
  <bpws:condition expressionLanguage="http://www.w3.org/TR/1999/REC-xpath-19991116">
     not( ( bpws:getVariableData("InternalOrdering_BlockEnterandApprove_OUT",
                                  "_STRUCT/OrderData/OrderStatus") != "postponed" ) )
     or not( bpws:getVariableData("InternalOrdering_BlockEnterandApprove_ST", "ActivityStatus") = "started" )
</bpws:condition>
…
</bpws:while>
```

<u>WID BPEL editor:</u>



Expression language: XPath 1.0

```
not( ( bpws:getVariableData("BlockEnterandApprove_OUT", "_STRUCT/OrderData/OrderStatus") != "postponed" ) ) or
not( bpws:getVariableData("BlockEnterandApprove_ST") = "started" )
```

Figure 61: Mapping an exit condition expression to BPEL

Activity expiration

Activity expiration can be referred to in FDL conditions and is therefore a crucial task for migration from WebSphere MQ Workflow. Figure 62 shows a simple example that illustrates how exit and transition conditions can use the expiration event of activity "Get customer name".

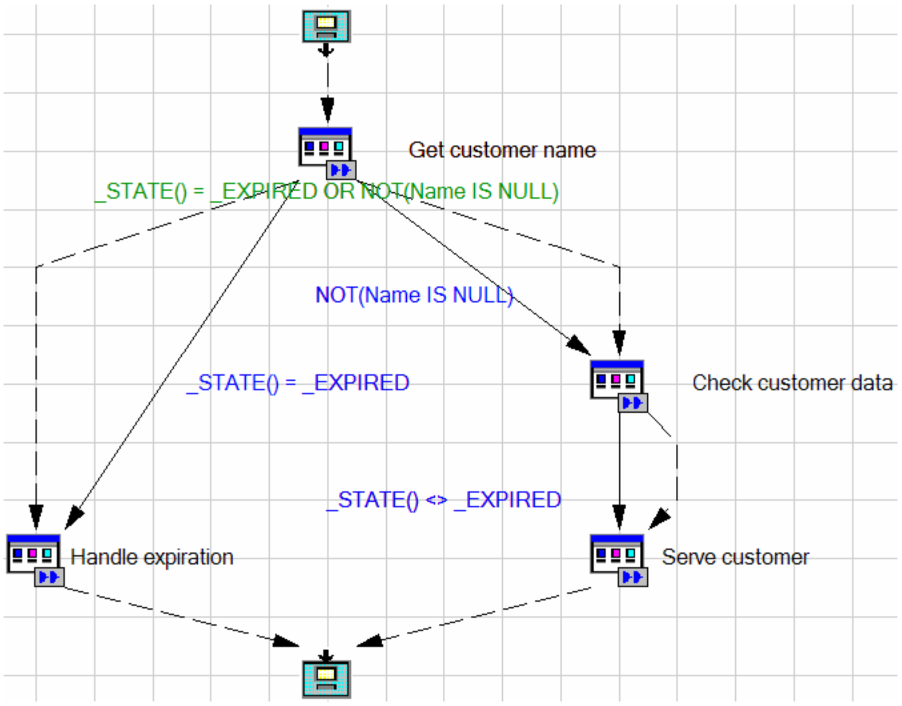


Figure 62: Activity expiration

In FDL you can specify expiration settings on a program activity that triggers the state "expired" on that node, which later on can be used in transition conditions. As the FDL2BPEL Conversion Tool uses "XPath" for the migrated condition expressions, you cannot directly refer to the corresponding "timeout" state in a BPEL process. That is why the FDL2BPEL Conversion Tool maps activity expiration in the following way:

1. Add a "Scope" activity and a "Sequence" activity to the "invoke" activity, if there is no exit condition.
2. Handle the "timeout" exception by adding a fault handler to:
   - the new "Scope" activity, if there is no exit condition
   - the "While Loop" activity, if there is an exit condition
3. Add an "Assign" activity to the fault handler which sets the "activity status" BPEL variable to the "expired" state (Note that the "activity status" variable is already introduced to map the "do until" semantics of an FDL block onto a BPEL "While Loop" activity).
4. Modify the translation of FDL conditions such that the "activity status" BPEL variable is queried in case of a "_state()" expression found in an FDL condition.
5. Example: the FDL transition condition "_STATE() = _EXPIRED" of some activity "Act1" becomes the BPEL link condition:

   bpws:getVariableData("Act1_ST") = "expired"

Figure 63 to Figure 66 show the translation of activities with and without an exit condition:

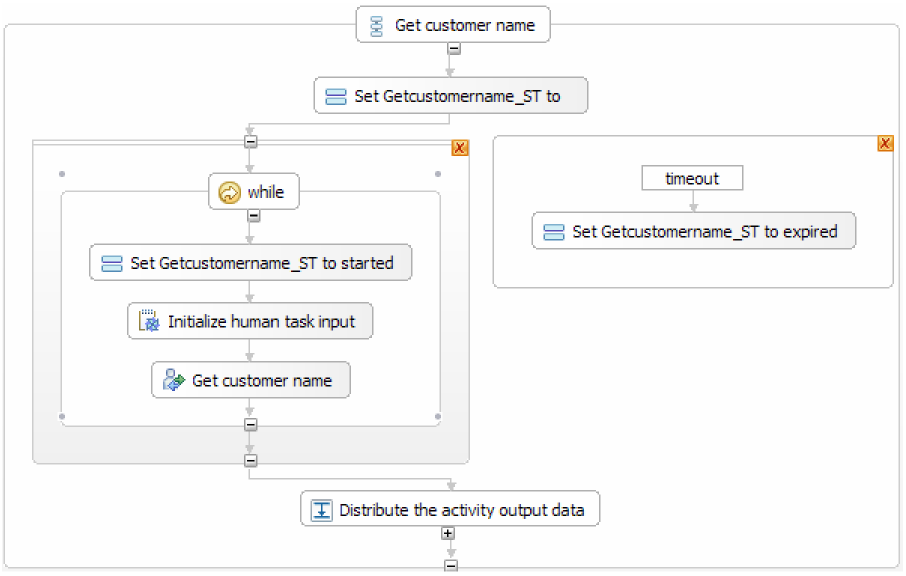
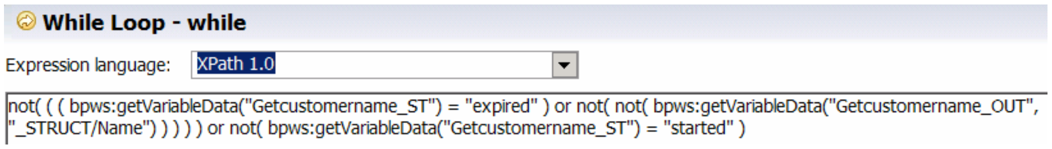

Get customer name
Set Getcustomername_ST to
while
Set Getcustomername_ST to started
Initialize human task input
Get customer name
timeout
Set Getcustomername_ST to expired
Distribute the activity output data

Figure 63: BPEL representation of FDL activity "Get customer name"


While Loop - while

Expression language: XPath 1.0

```
not( ( ( bpws:getVariableData("Getcustomername_ST") = "expired" ) or not( not( bpws:getVariableData("Getcustomername_OUT",
"_STRUCT/Name") ) ) ) ) or not( bpws:getVariableData("Getcustomername_ST") = "started" )
```

Figure 64: Translated exit condition of FDL activity "Get customer name"

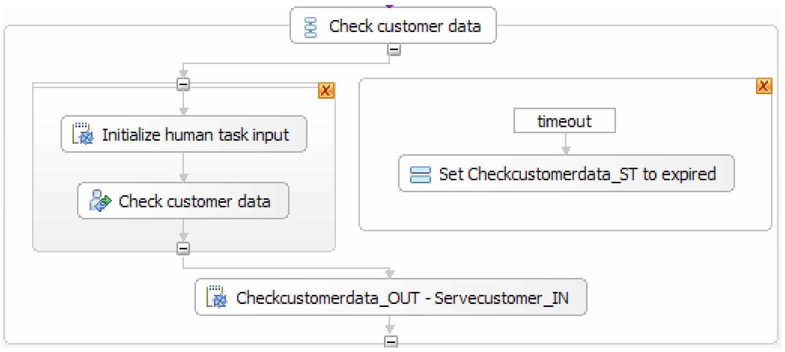
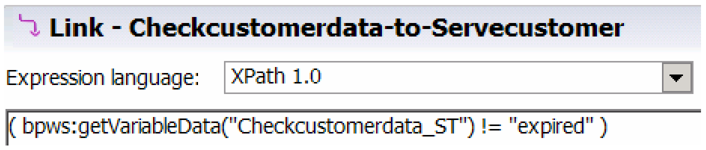

Check customer data
Initialize human task input
Check customer data
timeout
Set Checkcustomerdata_ST to expired
Checkcustomerdata_OUT - Servecustomer_IN

Figure 65: BPEL representation of FDL activity "Check customer data"


Link - Checkcustomerdata-to-Servecustomer

Expression language: XPath 1.0

```
( bpws:getVariableData("Checkcustomerdata_ST") != "expired" )
```

Figure 66: Translated transition condition refering to activity "Check customer data"

Mapping FDL data flow to BPEL

Unlike in FDL, you cannot use the concept of a *data connector* in BPEL. BPEL handles data by copying data from one variable to another using "Assign" activities or Java snippets. BPEL "Assign" activities contains <copy> instructions as child elements that are used to translate FDL data mapping instructions. In a similar way, you can also use Java snippets as an equivalent translation of FDL data mappings. In most cases, the FDL2BPEL Conversion Tool uses Java snippets, because the BPEL "Assign" activities do not have the same semantics of data mapping that you have in WMQWF.

The figure below shows a simple process in the diagram view of WMQWF Buildtime. It contains an activity "Act1" that passes data to the other activities "Act2" and "Act3":

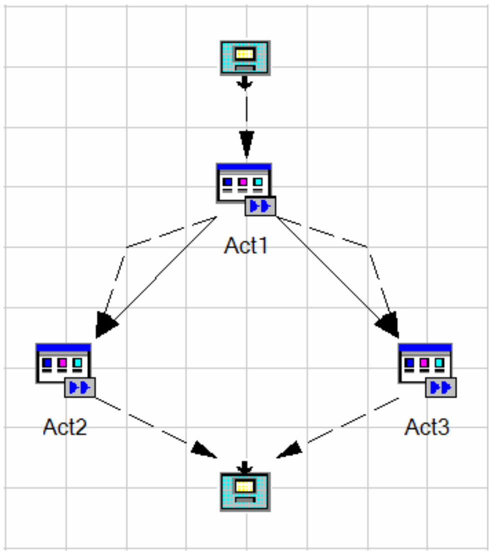


Figure 67: FDL data flow

Figure 68 shows part of the corresponding BPEL diagram view of the WID business process editor.

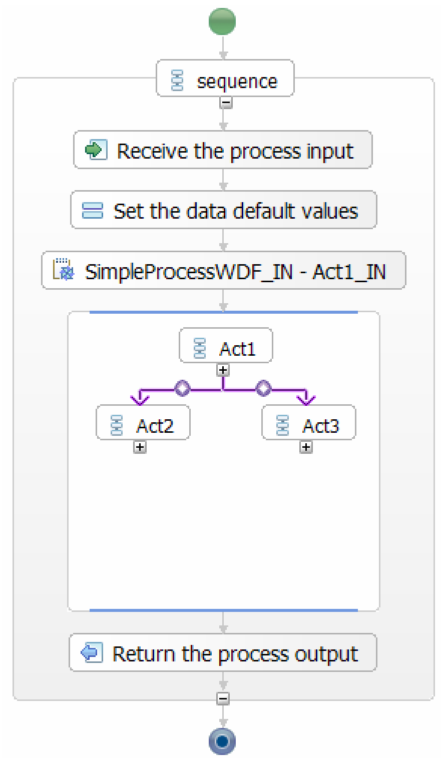

sequence
Receive the process input
Set the data default values
SimpleProcessWDF_IN - Act1_IN
Act1
Act2
Act3
Return the process output

Figure 68: BPEL representation of FDL data flow (1)

The icons for activities "Act1", "Act2", and "Act3" indicate collapsed BPEL structured activities of type "Sequence". For each FDL activity, the FDL2BPEL Conversion Tool generates a BPEL "Sequence" activity as a wrapper that consists of the corresponding translation of the activity itself (usually a BPEL "Invoke" activity) and other BPEL "Assign" activities and Java snippets that represent the data flow. The purpose of the "Sequence" construct is to ensure the correct execution order of data flow. In adition, using a "Sequence" activity as a wrapper helps to preserve the topology of the original WMQWF process model, which makes it is easier to verify the correctness of the migration.

The following figure shows the view of the same business process, but with an expanded "Sequence" activity generated for activity "Act1", which was classified as a "staff activity" (see "Activity classification rules" on page 60) and was therefore translated to a "human task":

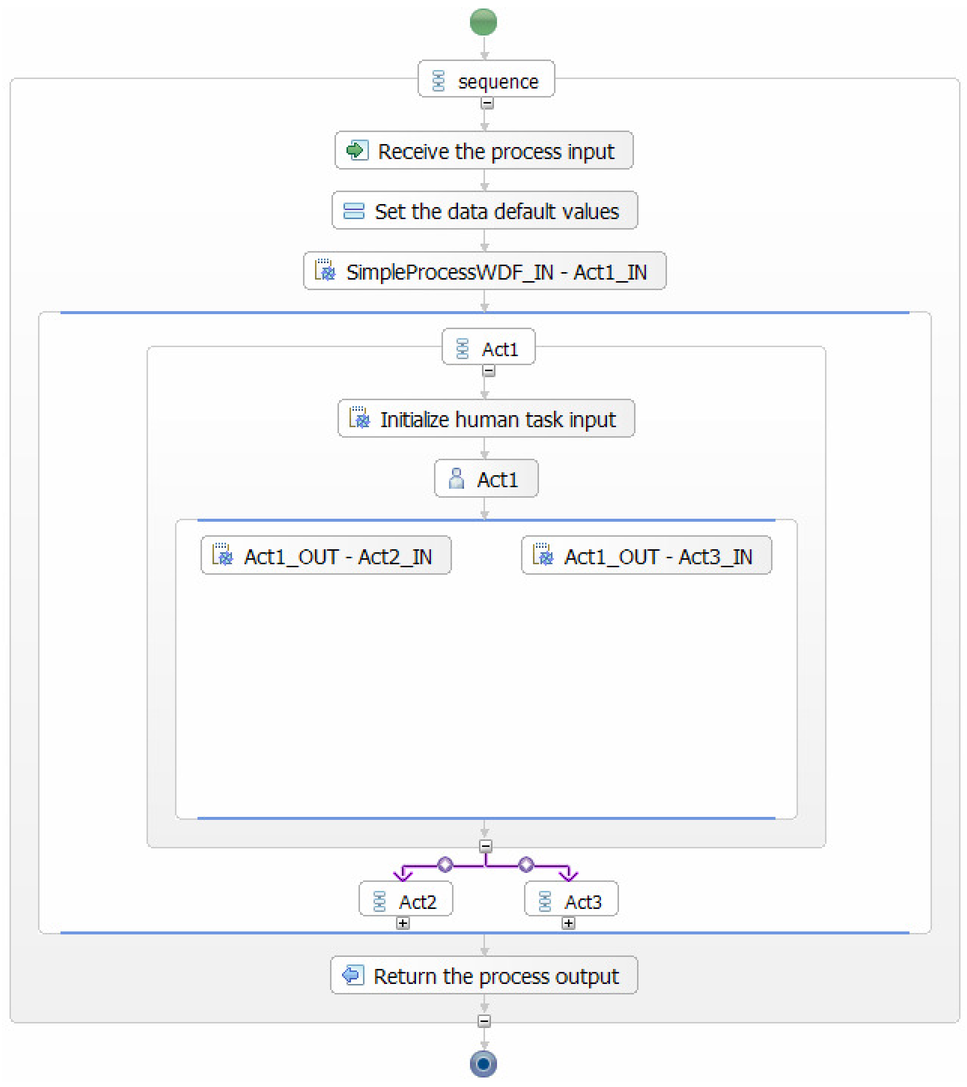

sequence
Receive the process input
Set the data default values
SimpleProcessWDF_IN - Act1_IN
Act1
Initialize human task input
Act1
Act1_OUT - Act2_IN
Act1_OUT - Act3_IN
Act2
Act3
Return the process output

Figure 69: BPEL representation of FDL data flow (2)

Starting from top of the diagram, the "Assign" activity titled "Set the data default values" initializes any default values for the process output data[57] and/or the data input and output of the activities contained in process "SimpleProcessWDF". The Java snippet ("SimpleProcessWDF_IN - Act1_IN") represents the translated data connector from the FDL "SOURCE" to the first activity "Act1" in the flow. The expanded BPEL "Sequence" for activity "Act1" contains

[57] See limitation "Initial process input data" on page 133.

a Java snippet that initializes the input BPEL variable of "Act1", preventing a "variable not set" exception if the inbound data flow is empty. Another "Parallel Activities" construct following the "Invoke" activity "Act1" consists of two Java snippets that are used as a translation for the outbound data connectors passing data to activities "Act2" and "Act3".

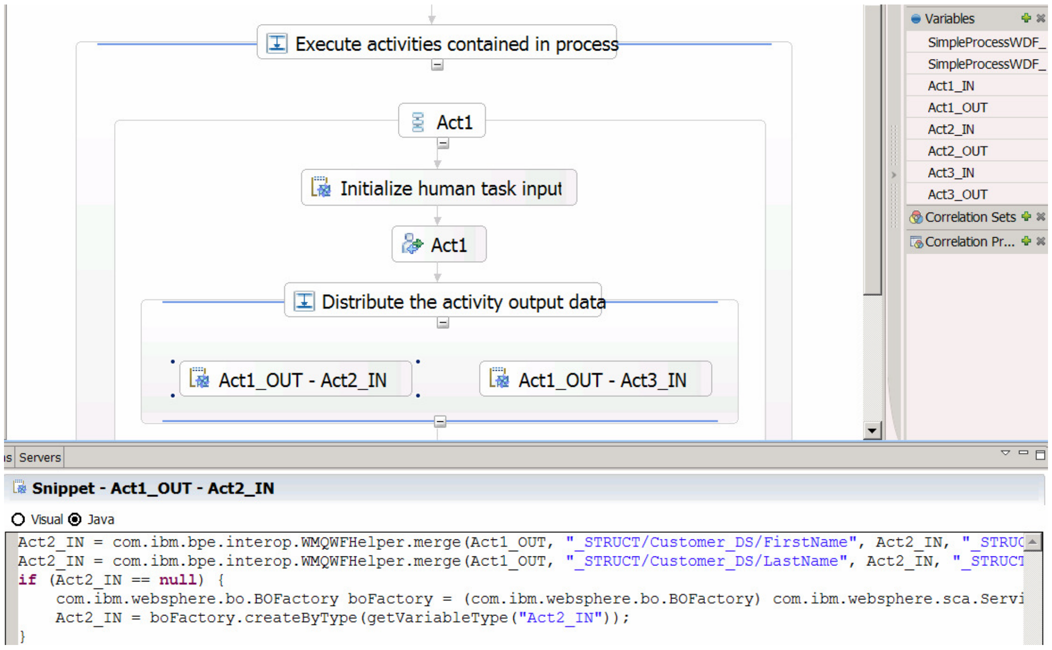
Consider an example mapping single data members:

FDL source code:

```
 DATA
   FROM 'Act1' TO 'Act2'
   MAP 'Customer_DS.FirstName' TO 'Customer_DS.FirstName'
   MAP 'Customer_DS.LastName' TO 'Customer_DS.LastName'
```

The corresponding implementation of this FDL data connector is a BPEL snippet with the name "Act1_OUT - Act2_IN":

BPEL editor (properties view of activity "Act1_OUT – Act2_IN"):



```
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/Customer_DS/FirstName", Act2_IN, "_STRUC
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/Customer_DS/LastName", Act2_IN, "_STRUCT
if (Act2_IN == null) {
    com.ibm.websphere.bo.BOFactory boFactory = (com.ibm.websphere.bo.BOFactory) com.ibm.websphere.sca.Servi
    Act2_IN = boFactory.createByType(getVariableType("Act2_IN"));
}
```

Figure 70: Mapping FDL data mapping to a Java snippet

The snippet contains the following Java instructions:

```
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(
                        SimpleProcessWDF_Act1_OUT,
                        "_STRUCT/Customer_DS/FirstName",
                        Act2_IN,
                        "_STRUCT/Customer_DS/FirstName",
                        getVariableType("Act2_IN"));
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(
                        Act1_OUT,
                        "_STRUCT/Customer_DS/LastName",
                        Act2_IN,
                        "_STRUCT/Customer_DS/LastName",
                        getVariableType("Act2_IN"));
if (Act2_IN == null){
   com.ibm.websphere.bo.BOFactory boFactory =
      (com.ibm.websphere.bo.BOFactory)
      com.ibm.websphere.sca.ServiceManager.INSTANCE.locateService(
                                     "com/ibm/websphere/bo/BOFactory");
   Act2_IN = com.ibm.websphere.bo.boFactory.createByType(
                                 getVariableType("Act2_IN"));
}
```

This Java code consists of two steps:

1. Perform the data mapping using an API called "*WMQWFHelper.merge()*" (highlighted in bold letters).
2. Initialize the target BPEL variable to prevent a "variable not set" exception, which would occur if the data mapping instructions refer to empty data members.

It is necessary to use the "*WMQWFHelper.merge()*" API to achieve the same data mapping semantics as in WMQWF. The "merge" API belongs to the "*com.ibm.bpe.interop*" package and has the following syntax:

public static DataObject	// target BPEL variable
merge(DataObject source,	// source BPEL variable
String sourcePath,	// XPath expression of source data member
DataObject target,	// target BPEL variable
String targetPath,	// XPath expression of target data member
Type targetType)	// type of target BPEL variable

The API is called "merge", because it does not just "overwrite"[58] the content of the target BPEL variable, but merges any preset data member of the target BPEL variable with new data values that are copied from the source BPEL variable. Table 8 summarizes the semantics of the "merge" operation:

[58] Note that BPEL "Assign" activities behave like that. If data merging is not required, such as for setting default values, the FDL2BPEL Conversion Tool only uses "Assign" activities.

Table 8: Semantic rules of "merge" operation

Source data member		Target data member	
Old value	**New value**	**Old value**	**New value**
valueA	valueA	valueB	valueA
valueA	valueA	<not set>	valueA
<not set>	<not set>	valueB	valueB
<not set>	<not set>	<not set>	<not set>

Notes:

- The above rules for merging data apply to elementary data types as well as to complex data types.
- Only data members that are specified by the XPath expression are changed. "Sibling" data members that belong to the same target BPEL variable will not be changed.
- If the target data member is changed according to the rules in the table, any "unset" ancestor data object in the hierarchy of a complex data object will be initialized before the "merge" operation is performed.

The following examples illustrate the data flow patterns of "data merging" and "data overwriting":

Table 9: Example 1 ("Merge" operation with two or more inbound data connectors)

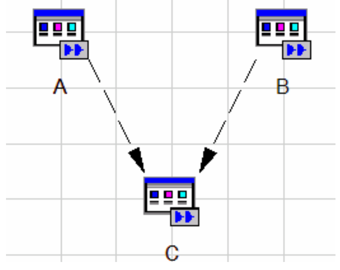
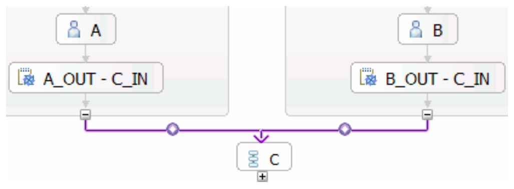
WMQWF process	BPEL process
	
Data is merged according to the principle "Last writer wins": If activity "B" executes after activity "A", then the input of "C" will be first set with the output data from "A" (snippet "A_OUT – C_IN" on the right). Subsequently, the output of "B" (snippet "B_OUT – C_IN" on the right) will update the input of "C" according to the semantic rules of the "merge" API.	

Table 10: Example 2 ("Merge" operation with inbound data connector leading to an input container with default values)

The data from the inbound data connector updates the default values of the activity input, such that the inbound data connector "wins".

Table 11: Example 3 ("Merge" operation with inbound data connector and data loop connector)

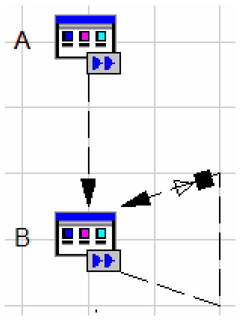
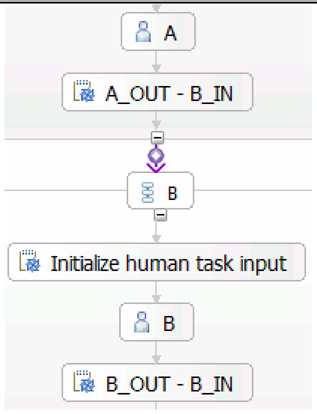
WMQWF process	BPEL process
	 A A_OUT - B_IN B Initialize human task input B B_OUT - B_IN
The input of activity "B" receives first data from the output of activity "A" (the snippet "A_OUT –B_IN"). Subsequently, activity "B" is executed and the output of "B" updates its own input data using the data loop connector (snippet "B_OUT – B_IN").	

Table 12: Example 4 (Combining examples 1 – 3)

The data is "merged" in the following sequence: 1. The activity input is set with default values. 2. The values from inbound data connector(s) update the activity input. 3. The activity output updates the activity input resulting from step 2.

Table 13: Example 5 (UPES or PEA[59] activity ("staff activity") with default output data and/or default data connector)

In this case, no data "merging" takes place. The activity result overwrites any preset data values in the output data container (in BPEL, the output variable).

[59] A PEA is a Program Execution Agent, which invokes a local program application. Migrating "PEA" activities is not supported (see limitation "FDL ProgramActivity" on page 142). The FDL2BPEL Conversion Tool translates them to an "assumed staff activity" (see "Activity classification rules" on page 60).

Table 14: Example 6 (Process activity with default output data and/or default data connector)

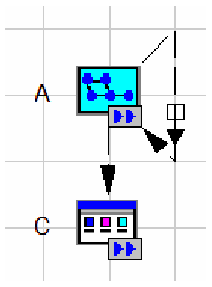
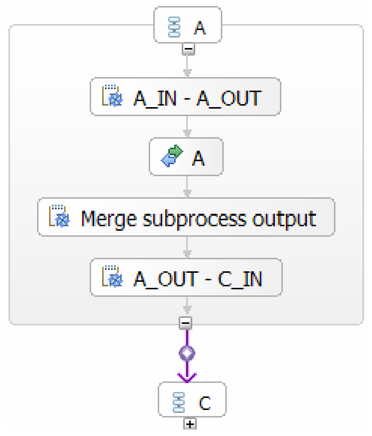
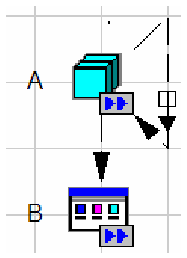
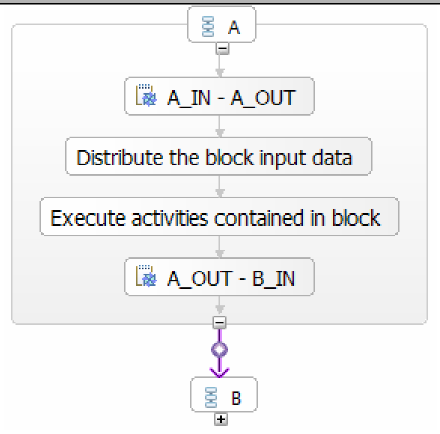
WMQWF process	BPEL process
	 A A_IN - A_OUT A Merge subprocess output A_OUT - C_IN C
First, the input of activity "A" updates the output of activity "A" using the default data connector (in BPEL, the snippet "A_IN – A_OUT"). Then, the result of the subprocess invocation of activity "A" updates the contents of its output container (in BPEL, the output variable "A_OUT"). In the BPEL model the last "merging" step is performed in an additional snippet "Merge subprocess output": The output of the subprocess is sent to an "auxiliary" BPEL variable "A_SP_OUT" and the snippet uses API "*WMQWFHelper.merge()*" to update the data in "A_OUT" with the data in variable "A_SP_OUT".	

Table 15: Example 7 (Block activity with default output data and/or default data connector)

WMQWF process	BPEL process
	 A A_IN - A_OUT Distribute the block input data Execute activities contained in block A_OUT - B_IN B
Similar to example 6, the input data of activity "A" updates the output data using the default data connector (in BPEL, the snippet "A_IN – A_OUT"). Then the input of "A" is distributed to the start activities inside block "A" (see the flow activity "Distribute the block input data, which contains respective snippets). Finally, the result of the block activity "A" updates the contents of its output container.	

Mapping WMQWF staff assignment criteria to TEL people assignment criteria

Introduction

WMQWF staff assignment criteria cannot be consistently mapped to corresponding TEL peope assignment criteria because of differences between the modelling constructs. The following differences require a compromise solution:

Table 16: Comparing staff/people assignment criteria (WMQWF vs. TEL)

WMQWF staff assignment criteria	TEL people assignment criteria
Except for block activities, staff assignment criteria can be assigned to any activity type.	• People assignment criteria can only be assigned to "staff activities". • "Staff activities" exclusively model the interaction with a user • "Staff activities" cannot model the invocation of a program or service.
Staff assignment criteria can be inherited. For example: Program activities without specific staff assignments may inherit staff assignment criteria from the process properties "Staff" page.	TEL people assignment criteria cannot be inherited from another context.
Staff assignments can result from combining multiple staff assignment criteria. Examples: • On the "Staff 2" page of the activity properties you can simultaneously specify staff assignments "Members of roles" and "Organization". At runtime, staff resolution results from a union set of assignment criteria. • There is an "Include process assignment" radio button on the "Control" page of the activity. If this option is selected, the organization and role settings for the process model are part of the final staff assignment of the activity.	TEL staff resolution always results from a single people assignment criterion.
WMQWF lets you specify staff assignment policies. Examples: • Prefer not absent users. • Assign substitute if user is absent. • Include members only. • Include reporting managers. • Include child organizations.	"Include reporting managers" <u>cannot</u> be mapped to TEL.
WMQWF lets you specify "Level" related staff assignment criteria.	"Level" is an unknown concept in TEL.

Mapping limitations

Using default people assignment criteria

Because of the differences listed in the above comparison, modelling staff assignment criteria with the TEL default "people assignment criteria" currently cannot achieve all possible WMQWF runtime staff resolution behaviors. The FDL2BPEL Conversion Tool offers a translation of WMQWF staff assignment criteria to TEL "default people assignment criteria" as specified in the "VerbSet.xml". You can map more complex staff assignment criteria by specifying customized TEL people assignment criteria. For instance, you can add a new definition of people assignment criteria that models the combination of two defaults "people assignment criteria". Nevertheless, be aware of the limited staff query inheritance capabilities of TEL and that a BPEL process model cannot be customized to achieve combined people assignment criteria or staff resolution strategies.

Distinction between "UPES activities" and "staff activities"

It is assumed that WMQWF "UPES" activities (program activities that are associated with a user-defined program execution server) are best mapped to non-interactive "invoke" activities that have a "partnerLink" assigned. The FDL2BPEL Conversion Tool maps all other WMQWF program activities to "empty activities" or "staff activities" according to the activity classification rules (see "Activity classification rules" on page 60). "Staff activities" are translated to BPEL "to-do tasks" with the "partnerLink" attribute set to "null", as described in the IBM BPEL extensions specification.

Migration of staff assignment definitions to be inherited

WebSphere Process Server does not have equivalent rules of inheritance for model properties like WebSphere MQ Workflow. In particular, migrating properties that your process inherits from domain or system definitions is not supported by the FDL2BPEL Conversion Tool, because its migration scope is limited to business processes, which should not depend on system migration aspects. The FDL2BPEL Conversion Tool translates the capability of WMQWF to inherit staff assignment definitions from the process settings at runtime into a sort of static inheritance at migration time:

Assume that your WMQWF process has an FDL "program activity" that fulfills the following conditions:

1. default staff assignment (for example, "all people")
2. mapping to BPEL will be a "staff activity" according to the "activity classification rules" (cp. page 60)
3. staff assignment "inherited" is selected

Then the FDL2BPEL Conversion Tool looks up the staff definition in process settings and maps those to equivalent TEL people assignment criteria for the activity (see "Mapping of staff assignment criteria" below).

Mapping of staff assignment criteria

Table 17 shows the mapping between WMQWF staff assignment criteria and TEL default people assignment criteria:

Table 17: Mapping WMQWF staff assignment criteria to TEL default people assignment criteria

WMQWF staff assignment criteria	TEL people assignment criteria
All people	<tel:name>Everybody</tel:name>
Staff from predefined members	**No mapping available**
People[60]	<tel:name>Users by user ID</tel:name> <tel:parameter id="UserID">*name or %XPath expression%*</tel:parameter> <tel:parameter id="AlternativeID1">*name or %XPath expression%*</tel:parameter> <tel:parameter id="AlternativeID2">*name or %XPath expression%*</tel:parameter>
Process administrator	<tel:name>Users by user ID</tel:name> <tel:parameter id="UserID"> *name or %XPath expression% or* %wf:process administrators% </tel:parameter>
Process starter	<tel:name>Users by user ID</tel:name> <tel:parameter id="UserID">%wf:process.starter%</tel:parameter>
Manager of process starter	<tel:name>Manager of Employee by user ID</tel:name> <tel:parameter id="EmployeeUserID"> %wf:process.starter%</tel:parameter>
Starter of activity	<tel:name>Users by user ID</tel:name> <tel:parameter id="UserID">%wf:activity(*Name*).owner%</tel:parameter>
Manager of starter of activity	<tel:name>Manager of Employee by user ID</tel:name> <tel:parameter id="EmployeeUserID"> %wf:activity(name).owner%</tel:parameter>
Exclude starter of activity	No mapping available
Members of roles[61]	<tel:name>Group Members</tel:name> <tel:parameter id="GroupName">*name or %XPath expression%*</tel:parameter> <tel:parameter id="IncludeSubGroups">false</tel:parameter> <tel:parameter id="AlternativeGroupName1">*name or %XPath expres-sion%*</tel:parameter> <tel:parameter id="AlternativeGroupName2">*name or %XPath expres-sion%*</tel:parameter>
Coordinator of role	No mapping available
Organization (strategy "include members only")	<tel:name> Group Members</tel:name> <tel:parameter id=" GroupName">*name*</tel:parameter> <tel:parameter id= IncludeSubGroups >false</tel:parameter>
Organization (strategy "include reporting managers")	No mapping available
Organization (strategy "include child organi-zations")	<tel:name> Group Members</tel:name> <tel:parameter id=" GroupName">*name*</tel:parameter> <tel:parameter id= IncludeSubGroups >true</tel:parameter>
Manager of organiza-tion	No mapping available

Notification

In MQ Workflow, the process modeler can specify a period of time in which an activity must finish. A designated person receives a notification work item if the activity is not completed in the specified time. If the notification is not completed within the specified time, a second notifi-cation may be given to the process administrator.

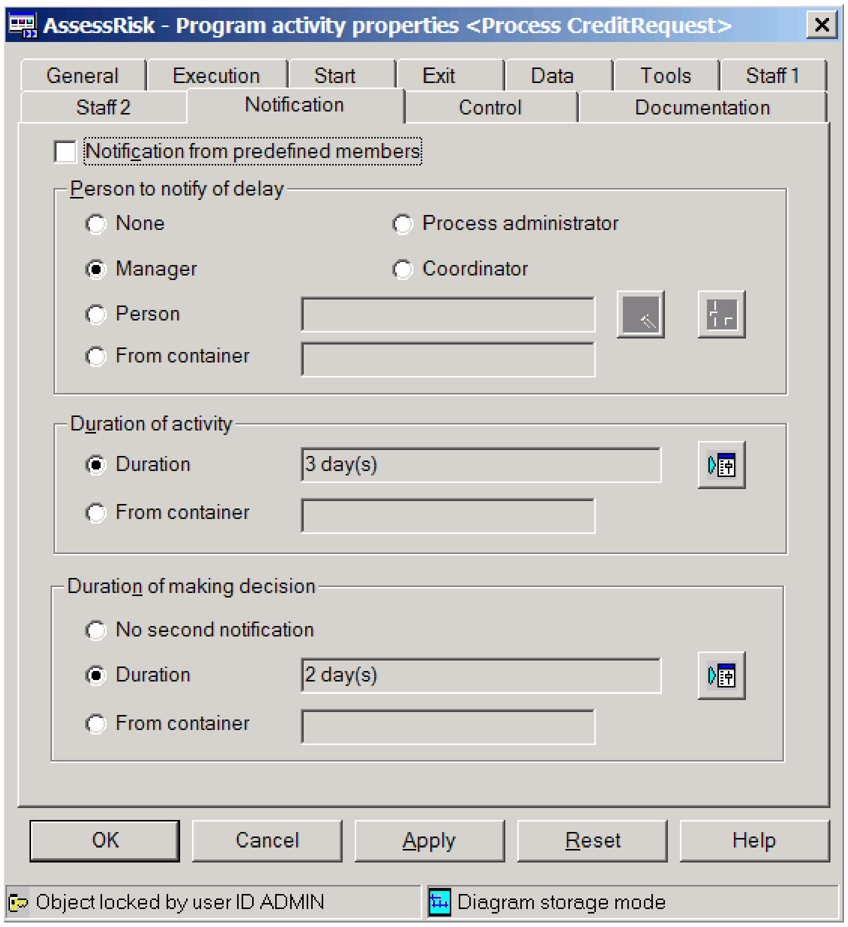
The FDL2BPEL Conversion Tool maps activity notification to an escalation for a human task in WebSphere Process Server. Note that you cannot migrate the notification of an activity that is translated to an activity type other than "staff activity", according to the FDL2BPEL activity classification rules (cp. section "Activity classification rules" on page 60). If you have specified

[60] Restricted to the first three items from the people list field.

[61] Restricted to the first three items from the role list field.

a second notification for an activity of your WMQWF model, the FDL2BPEL Conversion Tool will translate it to an "escalation chain".

As an example, Figure 71 below shows how you can specify in WMQWF Buildtime a person to notify of a delay, if the activity is not completed within three days. A second notification will be sent to the process administrator if the activity is still open after two more days.



☐ Notification from predefined members

Person to notify of delay: None [✓ Manager] Process administrator Coordinator Person From container

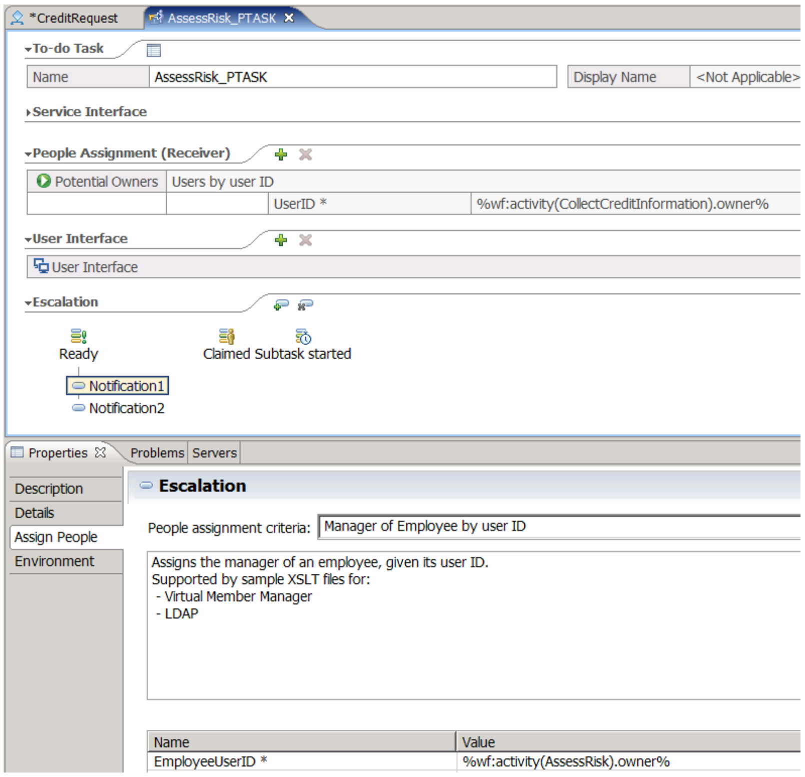
Duration of activity: [✓ Duration] 3 day(s); From container

Duration of making decision: No second notification [✓ Duration] 2 day(s); From container

Object locked by user ID ADMIN | Diagram storage mode

Figure 71: First and second notification of an FDL activity

The next two figures (Figure 72 and Figure 73) show how the first and second notifications are reflected in the WID human task editor as the members of an escalation chain:


*CreditRequest | AssessRisk_PTASK

To-do Task

Name	AssessRisk_PTASK	Display Name	<Not Applicable>

Service Interface

People Assignment (Receiver)

Potential Owners	Users by user ID	
	UserID *	%wf:activity(CollectCreditInformation).owner%

User Interface

User Interface

Escalation

Ready
Claimed Subtask started
Notification1
Notification2

Properties | Problems | Servers

Description
Details
Assign People
Environment

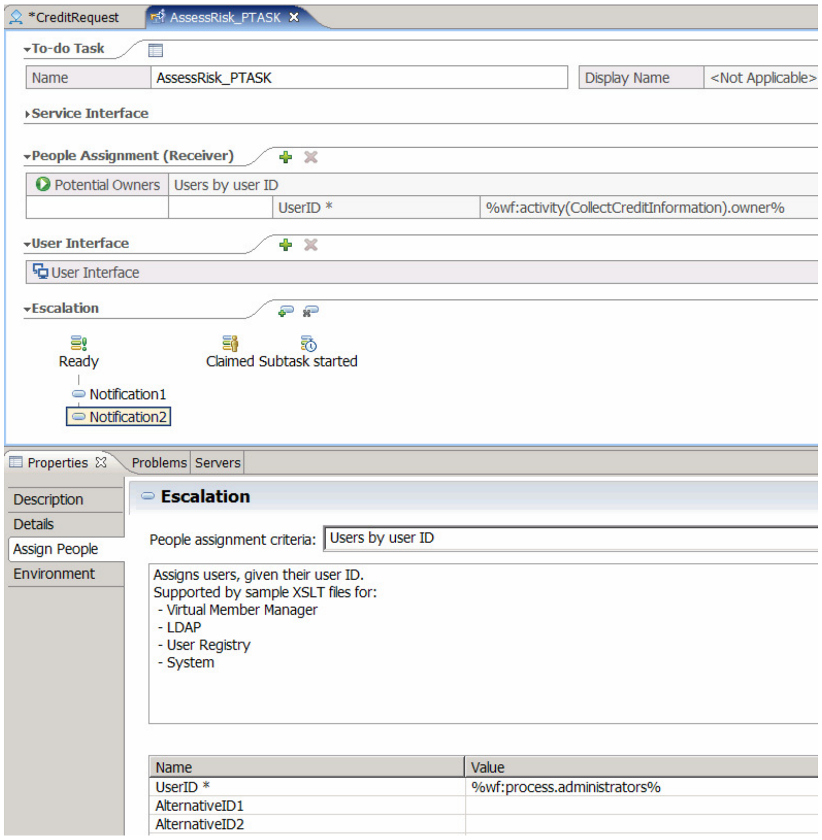
Escalation

People assignment criteria: Manager of Employee by user ID

```
Assigns the manager of an employee, given its user ID.
Supported by sample XSLT files for:
 - Virtual Member Manager
 - LDAP
```

Name	Value
EmployeeUserID *	%wf:activity(AssessRisk).owner%

Figure 72: Mapping first notification to an escalation step of a human task


*CreditRequest | AssessRisk_PTASK

To-do Task

Name	AssessRisk_PTASK	Display Name	<Not Applicable>

Service Interface

People Assignment (Receiver)

Potential Owners	Users by user ID	
	UserID *	%wf:activity(CollectCreditInformation).owner%

User Interface

User Interface

Escalation

Ready
Claimed Subtask started
Notification1
Notification2

Properties | Problems | Servers

Description
Details
Assign People
Environment

Escalation

People assignment criteria: Users by user ID

```
Assigns users, given their user ID.
Supported by sample XSLT files for:
 - Virtual Member Manager
 - LDAP
 - User Registry
 - System
```

Name	Value
UserID *	%wf:process.administrators%
AlternativeID1	
AlternativeID2	

Figure 73: Mapping second notification to an escalation step of a human task

Substitution

MQWF supports substitution by automatically transferring work items to another person when the person to whom an activity was originally assigned is declared as absent.

The FDL2BPEL Conversion Tool migrates the definitions for substitutes from WMQWF to WPS according to the following mapping rules for staff assignment policies:

Table 18: Substitution mapping rules

WMQWF Staff assignment control flag	TEL task property
Prefer not absent users	substitutionPolicy="SelectUserIfPresent"
Assign substitute if user absent	substitutionPolicy="SubstituteUserIfAbsent"

Mapping FDL to Service Component Architecture

WebSphere Business Process Choreographer supports the Service Component Architecture (SCA). Accordingly, the FDL2BPEL Conversion Tool generates "service component" and "service import" artifacts as files with the extensions ".component" and ".import". They contain XML code that conforms to the "Service Component Definition Language" (SCDL). WebSphere Integration Developer provides an "Assembly editor" that displays a graphical view of your business application in terms of SCA.

For example, the following figures show the WMQWF Buildtime diagrams of a "Nice Journey" WMQWF business process that contains a main process "Nice_Journey" and a subprocess "NJ_BookCar". Note that each of the program activities is implemented by a UPES according to the "Migration Best Practices" recommendations:

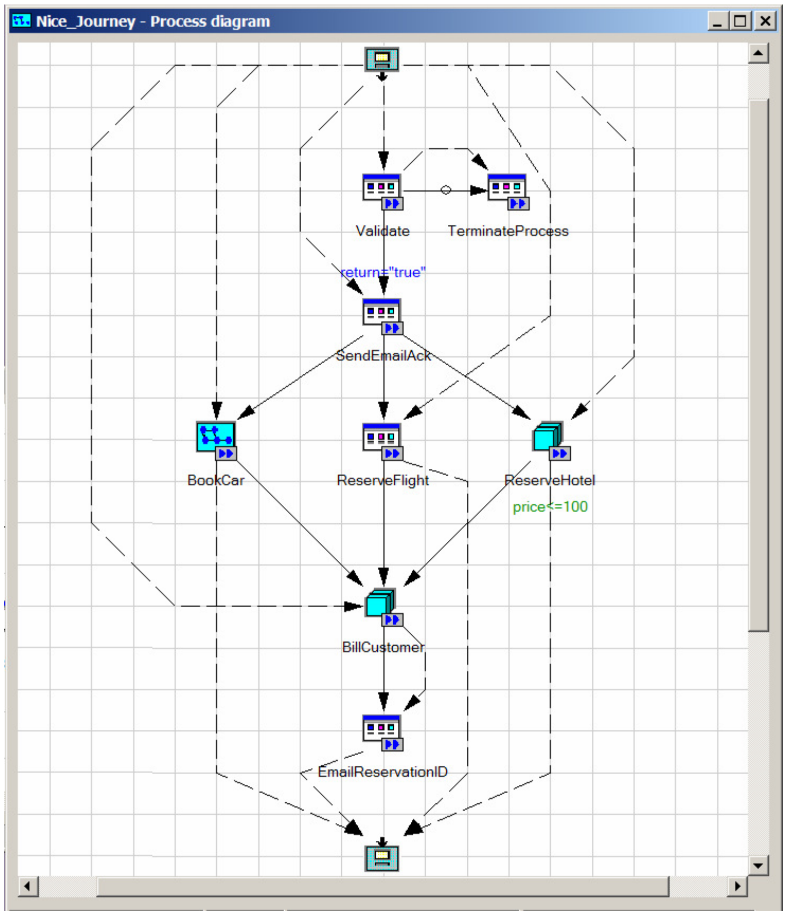


Figure 74: FDL process "Nice Journey"

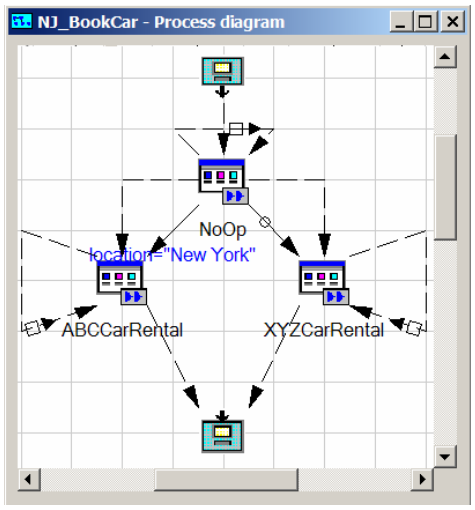


Figure 75: FDL subprocess "Book Car"

The next figure shows the same processes after they have been migrated and imported into WID in the "assembly diagram" view. The processes now "implement" the SCA components "Nice_Journey" and "NJ_BookCar", which in turn are "wired" to several "service import" icons that "implement" the necessary UPES applications:

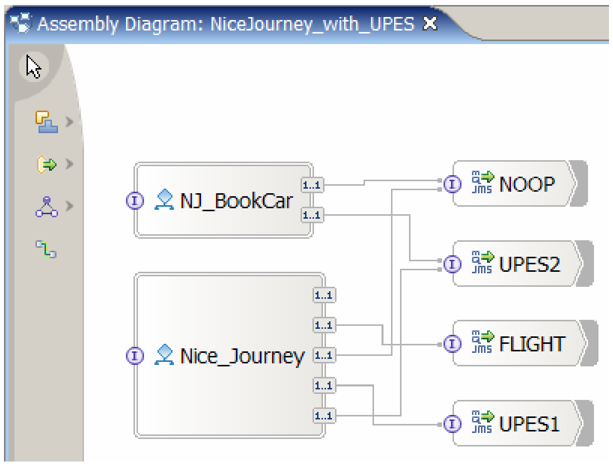


Figure 76: WID assembly diagram view

For more information about the content of the generated files, refer to the corresponding SCA literature in the WebSphere Information Center.

SCA Components

For each process definition in the FDL file, the FDL2BPEL Conversion Tool generates a single SCA component artifact that is implemented by a corresponding BPEL business process. For example, in the above assembly editor diagram view, the icons for the components "Nice-Journey" and "NJ_BookCar".

As previously mentioned in "Mapping FDL PROCESS_ACTIVITY to BPEL" on page 68, the FDL2BPEL Conversion Tool supports subprocess invocation by process template names (static or late bound), which conforms to the WMQWF runtime behaviour. This means that an explicit "wiring" between a subprocess "reference" hot spot of a "parent" SCA component and the "interface" hot spot of the respective subprocess SCA component is not needed in the assembly diagram. **Note: Do not try to add a supposed missing wire!** For example: in the assembly diagram below (Figure 77) you should not add a wire from the unwired WSDL reference hot spot of component "Nice_Journey" to the interface hot spot of component "NJ_BookCar".

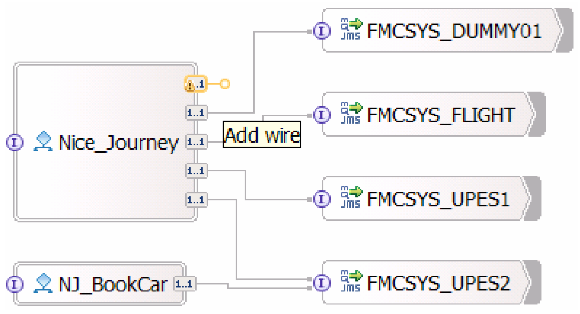


Figure 77: A deliberately omitted wire in the assembly diagram

SCA Imports

For each server definition in the FDL file, the FDL2BPEL Conversion Tool generates a single SCA import artifact, if and only if it is referred to in some FDL program activity definition. For example, the icons for the SCA imports "FMCSYS_DUMMY01", "FMCSYS_FLIGHT", "FMCSYS_UPES1", and "FMCSYS_UPES2" in the above assembly editor diagram view.

Data binding type

The "MQ JMS data binding" performs the transformation of data that is passed between a WPS business process executed by the WebSphere Business Process Choreographer and an MQ JMS messaging system. As mentioned in "Mapping the data exchanged with a UPES" on page 41, the communication with a UPES requires business objects that conform to a dedicated message type. These objects are serialized as an outbound XML text message type "***ActivityImplInvoke***"[62] and deserialized from XML text message types "***ActivityImplInvokeResponse***", according to the conventions of the WMQWF XML message interface (see "Part 5. The XML message interface" in IBM WebSphere MQ Workflow "Programming Guide" Version 3.6).

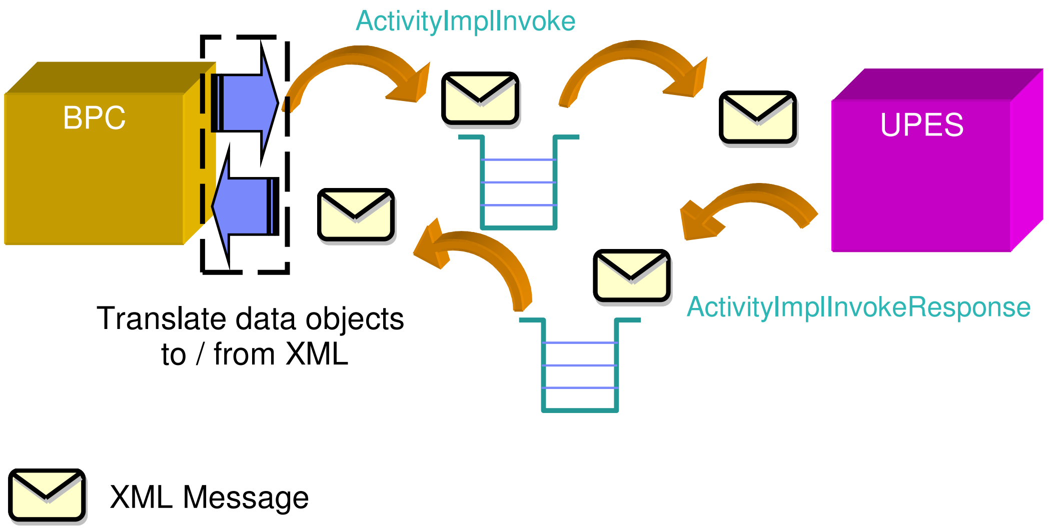

ActivityImplInvoke
BPC
UPES
ActivityImplInvokeResponse
Translate data objects to / from XML
XML Message

Figure 78: How BPC communicates with a UPES

[62] See limitation "XML message types" on page 140.

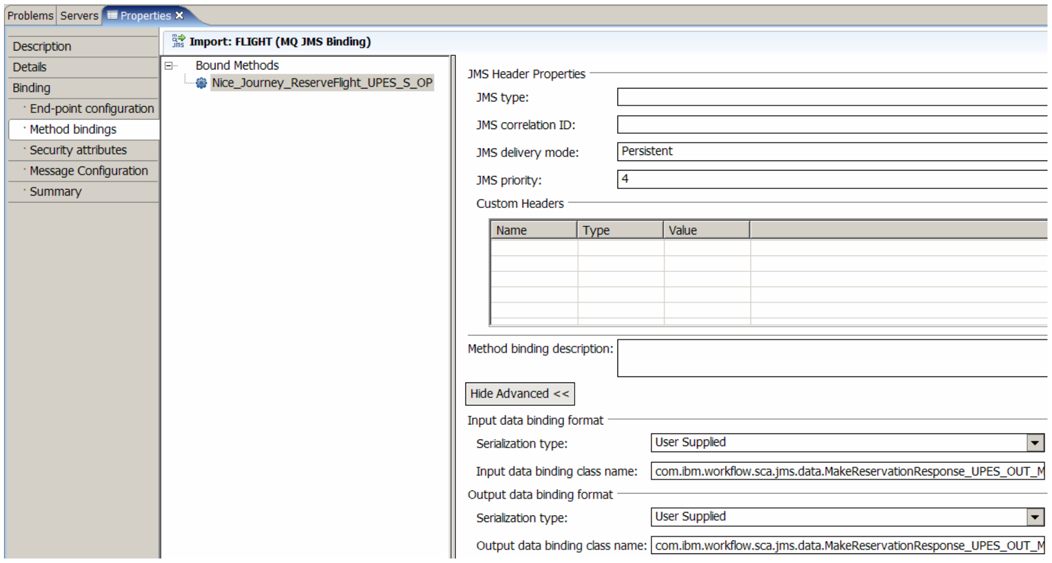
The FDL2BPEL Conversion Tool generates an MQJMS data binding (SCA "import") with an appropriate data binding type (serialization type).

The following XML code illustrates one of the above SCA imports ("FMCSYS_Flight"). The "data binding" type of the "makeReservationResponse" message is emphasized with under-lined letters:

XML source code of assembly diagram:

```
<?xml version="1.0" encoding="UTF-8"?>
<scdl:import
            xmlns:scdl="http://www.ibm.com/xmlns/prod/websphere/scdl/6.0.0"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:mqjms="http://www.ibm.com/xmlns/prod/websphere/scdl/mqjms/6.0.0"
            xmlns:ns1="http://www.ibm.com/xmlns/prod/websphere/mqwf/wsdl/"
            xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsdl/6.0.0"
            name="FMCSYS_FLIGHT"
            displayName ="FMCSYS_FLIGHT">
  <interfaces>
    <interface xsi:type="wsdl:WSDLPortType" portType="ns1:FMCSYS_FLIGHT_PT" preferredInteractionStyle="async">
      <method name="ReserveFlight_UPES"/>
    </interface>
  </interfaces>
  <esbBinding xsi:type="mqjms:MQJMSImportBinding" responseCorrelationScheme="RequestMsgIDToCorrelID">
    <outboundConnection>
      <mqConfiguration>
        <queueManager>FMCQM</queueManager>
      </mqConfiguration>
    </outboundConnection>
    <responseListener/>
    <send type="javax.jms.Queue" targetClient="JMS">
      <baseName>JMS_FLIGHT</baseName>
    </send>
    <receive type="javax.jms.Queue" targetClient="JMS">
      <baseName>JMS_FLIGHT_REPLY</baseName>
    </receive>
    <methodBinding
            method="ReserveFlight_UPES"
            inDataBindingType="com.ibm.workflow.sca.jms.data.ReserveFlight_Response_UPES_MSG_DataBindingImpl"
            outDataBindingType="com.ibm.workflow.sca.jms.data.ReserveFlight_Response_UPES_MSG_DataBindingImpl"
    />
  </esbBinding>
</scdl:import>
```

When the "FMCSYS_FLIGHT" SCA import is selected in the WID assembly editor you see the same data binding type on the Binding page of the properties view:



Import: FLIGHT (MQ JMS Binding)

Bound Methods
Nice_Journey_ReserveFlight_UPES_S_OP

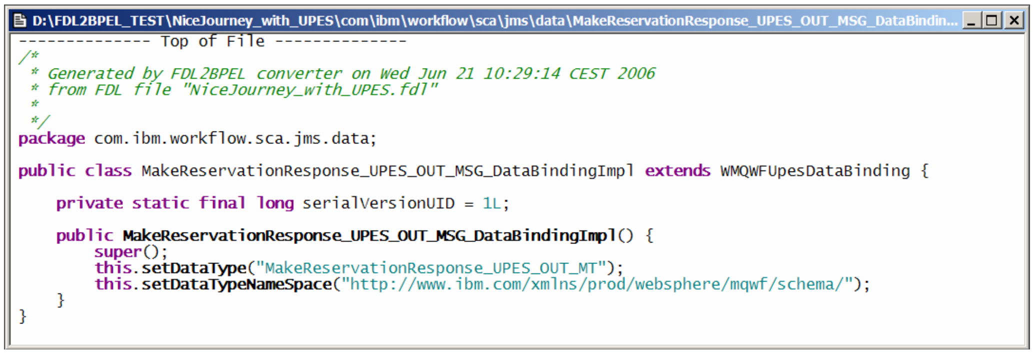
JMS Header Properties
JMS type:
JMS correlation ID:
JMS delivery mode: Persistent
JMS priority: 4
Custom Headers

Method binding description:

Input data binding format
Serialization type: User Supplied
Input data binding class name: com.ibm.workflow.sca.jms.data.MakeReservationResponse_UPES_OUT_M
Output data binding format
Serialization type: User Supplied
Output data binding class name: com.ibm.workflow.sca.jms.data.MakeReservationResponse_UPES_OUT_M

Figure 79: MQ JMS binding of an SCA import

The source code of the corresponding "user supplied" Java class "MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl" is also generated by the FDL2BPEL Conversion Tool:



```
D:\FDL2BPEL_TEST\NiceJourney_with_UPES\com\ibm\workflow\sca\jms\data\MakeReservationResponse_UPES_OUT_MSG_DataBindin...
-------------- Top of File --------------
/*
 * Generated by FDL2BPEL converter on Wed Jun 21 10:29:14 CEST 2006
 * from FDL file "NiceJourney_with_UPES.fdl"
 *
 */
package com.ibm.workflow.sca.jms.data;

public class MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl extends WMQWFUpesDataBinding {

    private static final long serialVersionUID = 1L;

    public MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl() {
        super();
        this.setDataType("MakeReservationResponse_UPES_OUT_MT");
        this.setDataTypeNameSpace("http://www.ibm.com/xmlns/prod/websphere/mqwf/schema/");
    }
}
```

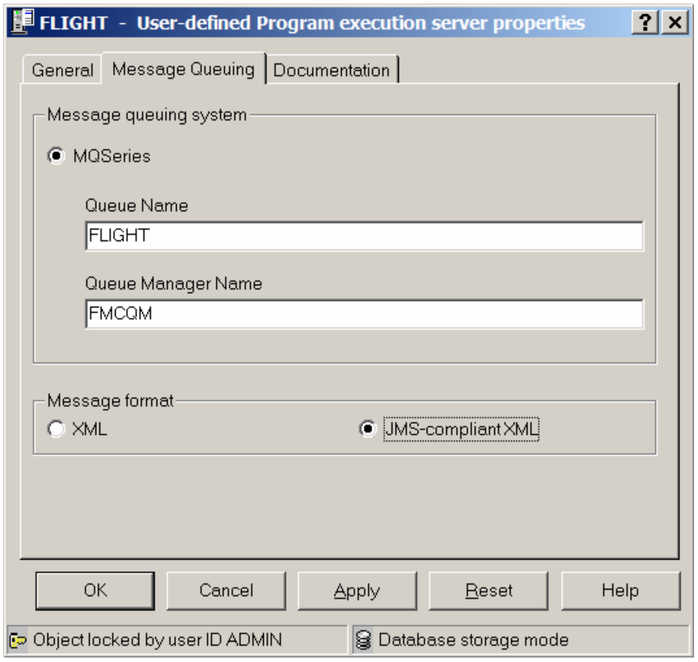
Figure 80: Java class implementing the data binding

The translation from the received XML representation of the received "MakeReservationResponse" message and the corresponding Java object for the "MakeReservationResponse" business object is done by the class "WMQWFUpesDataBinding", from which the class "MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl" is derived (note the "extends" keyword).

Because the "WMQWFUpesDataBinding" needs to know the translation target data type, "MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl" serves as a helper class to provide this piece of information.

Endpoint configuration

To be able to communicate with a UPES you also need to specify the messaging queues and a queue manager. The queue manager name is taken from the FDL server definition.



FLIGHT - User-defined Program execution server properties

Message queuing system

(● MQSeries)

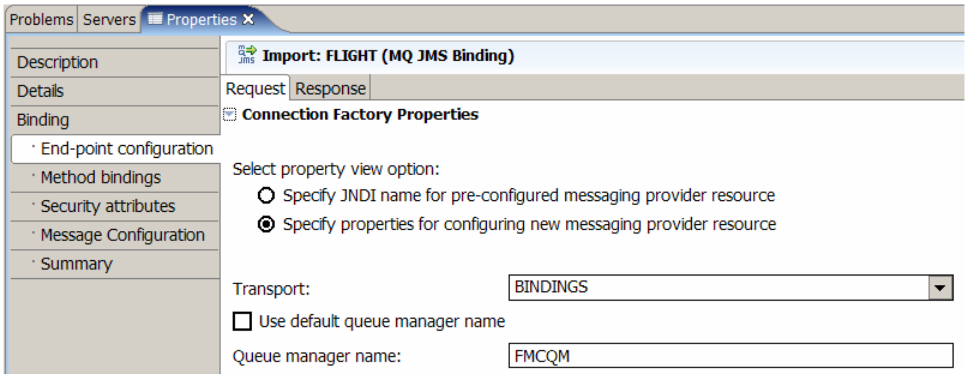
Queue Name: FLIGHT

Queue Manager Name: FMCQM

Message format: XML [✓ JMS-compliant XML]

Object locked by user ID ADMIN | Database storage mode

Figure 81: Queue manager in WMQWF Buildtime



Import: FLIGHT (MQ JMS Binding)

Connection Factory Properties

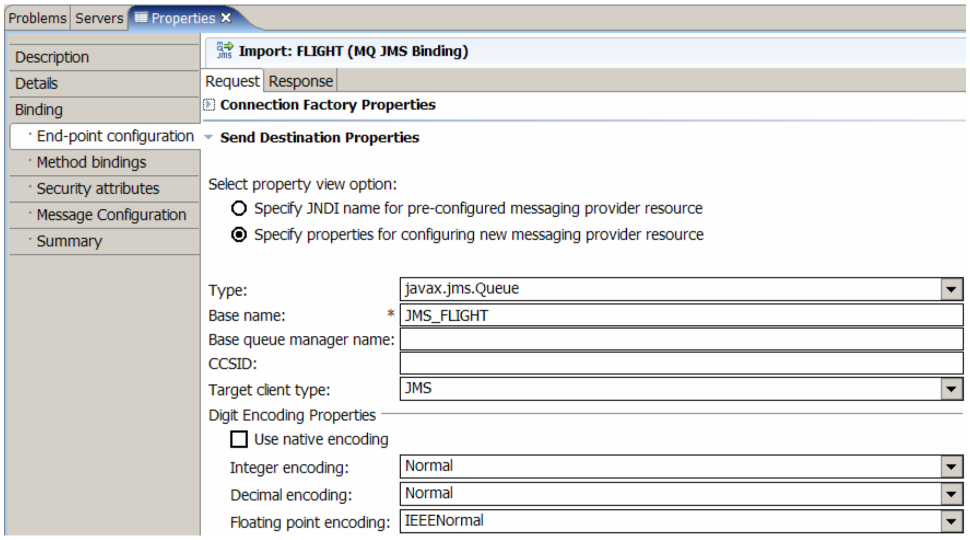
Select property view option:
( ) Specify JNDI name for pre-configured messaging provider resource
(●) Specify properties for configuring new messaging provider resource

Transport: BINDINGS

☐ Use default queue manager name

Queue manager name: FMCQM

Figure 82: Queue manager in the WID assembly editor (properties view)



Import: FLIGHT (MQ JMS Binding)

Connection Factory Properties

Send Destination Properties

Select property view option:
( ) Specify JNDI name for pre-configured messaging provider resource
(●) Specify properties for configuring new messaging provider resource

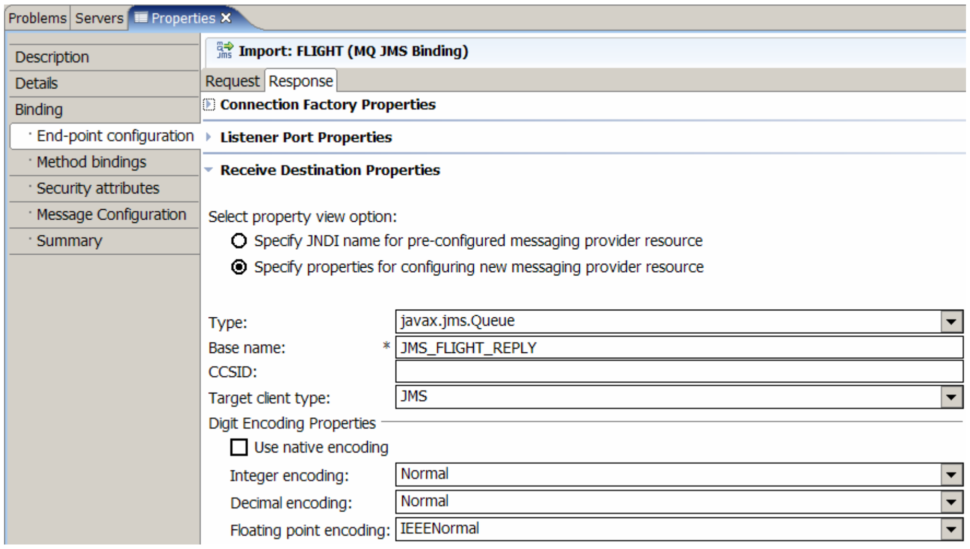
Type:	javax.jms.Queue
Base name: *	JMS_FLIGHT
Base queue manager name:	
CCSID:	
Target client type:	JMS

Digit Encoding Properties

☐ Use native encoding

Integer encoding:	Normal
Decimal encoding:	Normal
Floating point encoding:	IEEENormal

Figure 83: Request queue in the WID assembly editor (properties view)



Import: FLIGHT (MQ JMS Binding)

Connection Factory Properties

Listener Port Properties

Receive Destination Properties

Select property view option:
( ) Specify JNDI name for pre-configured messaging provider resource
(●) Specify properties for configuring new messaging provider resource

Type:	javax.jms.Queue
Base name: *	JMS_FLIGHT_REPLY
CCSID:	
Target client type:	JMS

Digit Encoding Properties

☐ Use native encoding

Integer encoding:	Normal
Decimal encoding:	Normal
Floating point encoding:	IEEENormal

Figure 84: Response queue in the WID assembly editor (properties view)

Note that the queue names are generated according to the following convention:

Table 19: Queue naming conventions

Queue type	Message format	Base name	Example
Request queue	XML	*<FDL queue name>*	"FLIGHT"
	JMS-compliant XML	JMS_*<FDL queue name>*	"JMS_FLIGHT"
Response queue	XML	*<FDL queue name>*_REPLY	"FLIGHT_REPLY"
	JMS-compliant XML	JMS_*<FDL queue name>*_REPLY	"JMS_FLIGHT_REPLY"

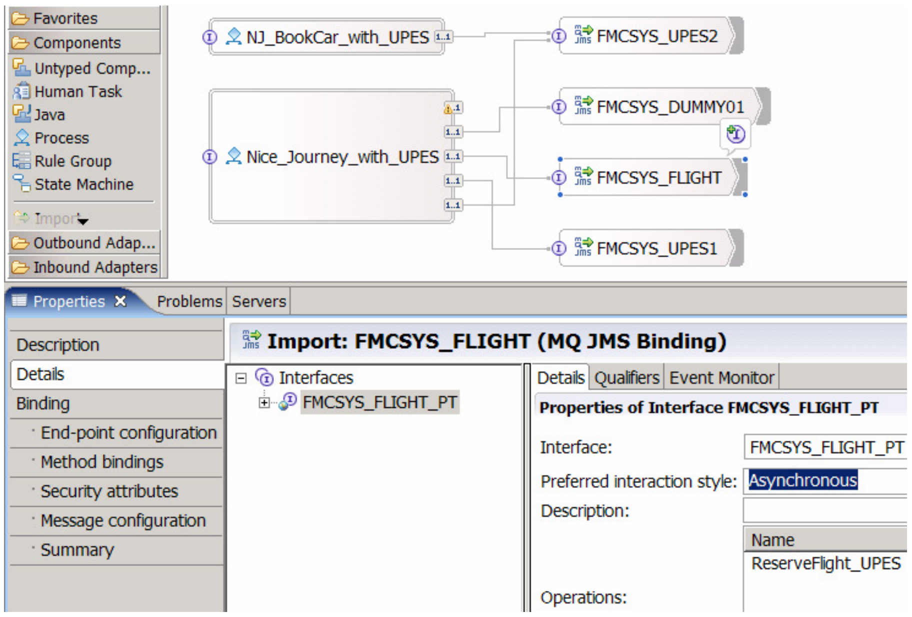
Preferred interaction style

On the Execution page of the settings notebook of a WMQWF "program activity" you can select the UPES execution mode, as follows:

Table 20: UPES execution mode

Execution mode	FDL property "SYNCHRONIZATION"	Explanation
Synchronous	**NESTED**	If selected, IBM WebSphere MQ Workflow waits for a response. The activity is started and receives the status running. The status is set to executed, when IBM WebSphere MQ Workflow is notified that the activity has finished.
Asynchronous	**CHAINED**	If selected, IBM WebSphere MQ Workflow continues without waiting. The activity runs asynchronous. It is started and, at the same time, the activity receives the status finished.

The FDL "SYNCHRONIZATION" property is mapped to the corresponding SCA import property "Preferred interaction style". In order to validate the property setting in WID, open the assembly diagram, select the respective "Import" of your UPES. On the "Properties" pane, click on the Details page, and select the only "interface" that is visible below the root of the "Interfaces" tree view. On the right side, make sure that the Details page which shows the value of property "Preferred interaction style" is visible (see Figure 85 below).



Import: FMCSYS_FLIGHT (MQ JMS Binding)

Properties of Interface FMCSYS_FLIGHT_PT

Interface: FMCSYS_FLIGHT_PT

Preferred interaction style: Asynchronous

Description:

ReserveFlight_UPES

Figure 85: Preferred interaction style of UPES interface

Chapter 5: WMQWF UPES migration details

Both XML-based and JMS-compliant XML-based UPES applications are supported. You can reuse UPES applications without code modifications that use only the following WMQWF message types:

- ActivityImplInvoke
- ActivityImplInvokeResponse

UPES applications that use or depend on other WMQWF message types, such as "ActivityExpired" or "TerminateProgram" need some reworking, because these messages are not supported in WPS.

During UPES migration you should check whether there are UPES applications that process messages which contain platform dependant "implementation data" (Note that WPS process will send "ActivityImplInvoke" messages without the element "ImplementationData").

For every UPES application that you want to reuse in a migrated WMQWF process, take into account that the MQJMS support in WebSphere Process Server has a message correlation mechanism that is different from the message correlation mechanism of a WMQWF server:

- A WMQWF server correlates response messages by means of the WMQWF *correlation id* inside the message payload. The *"message id"* and *"correlation id"* fields in the WebSphere MQ message descriptor (MQMD) are not used.
- WebSphere Process Server uses the MQMD fields "*message id"* and "*correlation id"* as described in the WebSphere MQ documentation and ignores the respective fields in the message payload.

To ensure correct message correlation, you might have to change your UPES application, as follows.

- "*XML"* based UPES applications must copy the content of "*message id"* field from the MQMD of the request message into the "*correlation id"* field of the MQMD of the response message.
- JMS-compliant XML-based UPES applications: Copy the *"message id"* from the request message into the *"correlation id"* field of the RFH2 header of the response message.

We recommend using the WMQWF UPES Framework V3.6 SP4. The UPES Framework handles the complete communication effort in the required way.

In most cases, you must create a new response queue for each UPES application because, other than in a WMQWF environment, the UPES response message is not sent back to the WMQWF execution server (that is the **EXEXMLINPUTQ** queue)[63].

[63] Make sure that your UPES application does not send the response message to the **EXEXMLINPUTQ** queue any time. The target queue information of the UPES response message is contained in the header of the UPES request message and must be taken from there.

Chapter 6: Optimizing the migrated business model

Optimizing BPEL process models

The generated BPEL files do not always represent the most compact translation of a given WMQWF process.

The FDL2BPEL Conversion Tool applies a translation pattern that preserves the original process topology. The advantage of this approach is that it makes it easier for you to become familiar with the generated BPEL process, because you can use the diagrams of the original FDL process model as a reference map. The drawback of this approach is that it will sometimes result in suboptimal solutions.

For example, simple FDL activity chains are translated to BPEL processes with redundant structured activities, like "Sequence" and "Parallel Activities".

Another example is a data connector that passes data unchanged from one activity to another one are sometimes better mapped to a shared BPEL variable where the FDL2BPEL Conversion Tool generates two BPEL variables and a snippet activity that does the data mapping.

You might want to rework a generated BPEL model in order to optimize it. For instance, you can remove redundant structured activities.

Consider the following WMQWF process with a simple chain of activities:

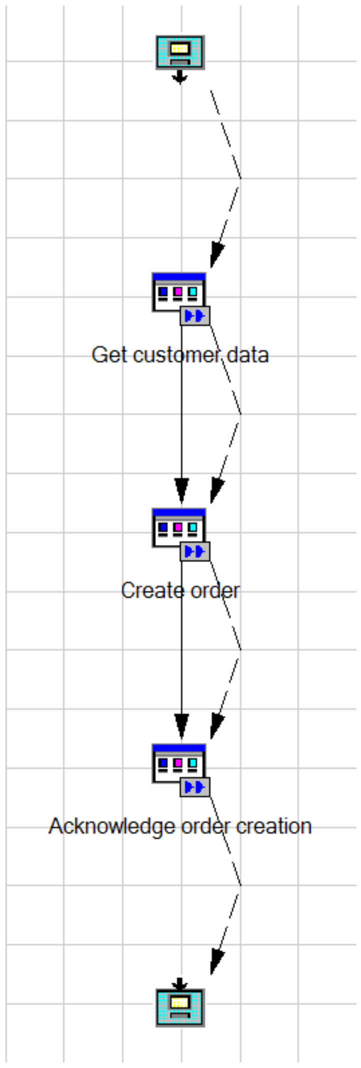


Figure 86: Simple chain of FDL activities

The FDL2BPEL Conversion Tool translates this into the following BPEL process:

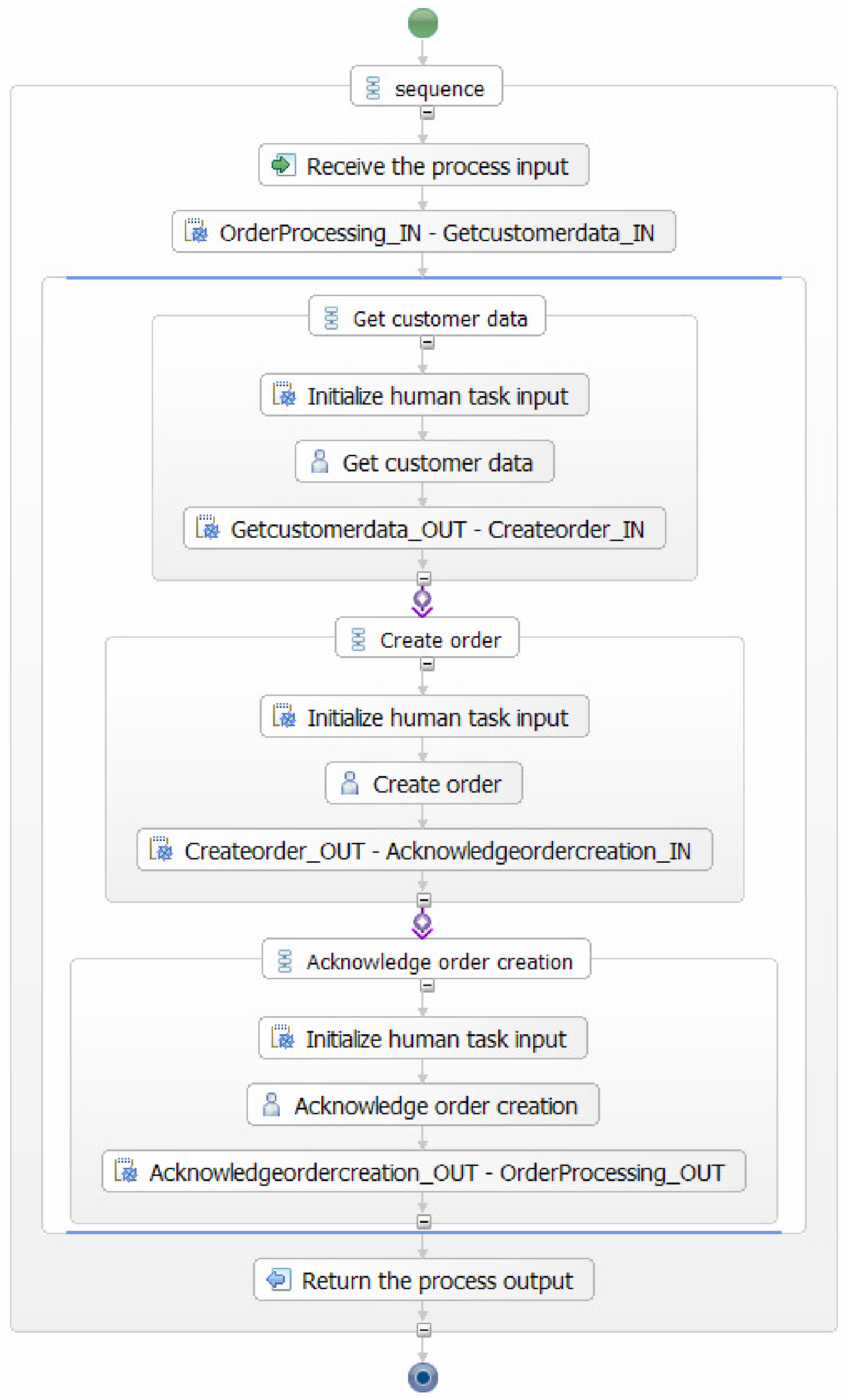

sequence
Receive the process input
OrderProcessing_IN - Getcustomerdata_IN
Get customer data
Initialize human task input
Get customer data
Getcustomerdata_OUT - Createorder_IN
Create order
Initialize human task input
Create order
Createorder_OUT - Acknowledgeordercreation_IN
Acknowledge order creation
Initialize human task input
Acknowledge order creation
Acknowledgeordercreation_OUT - OrderProcessing_OUT
Return the process output

Figure 87: BPEL activity chain with redundant structured activities

Removing the redundant structured activities would result in the following equivalent BPEL process:

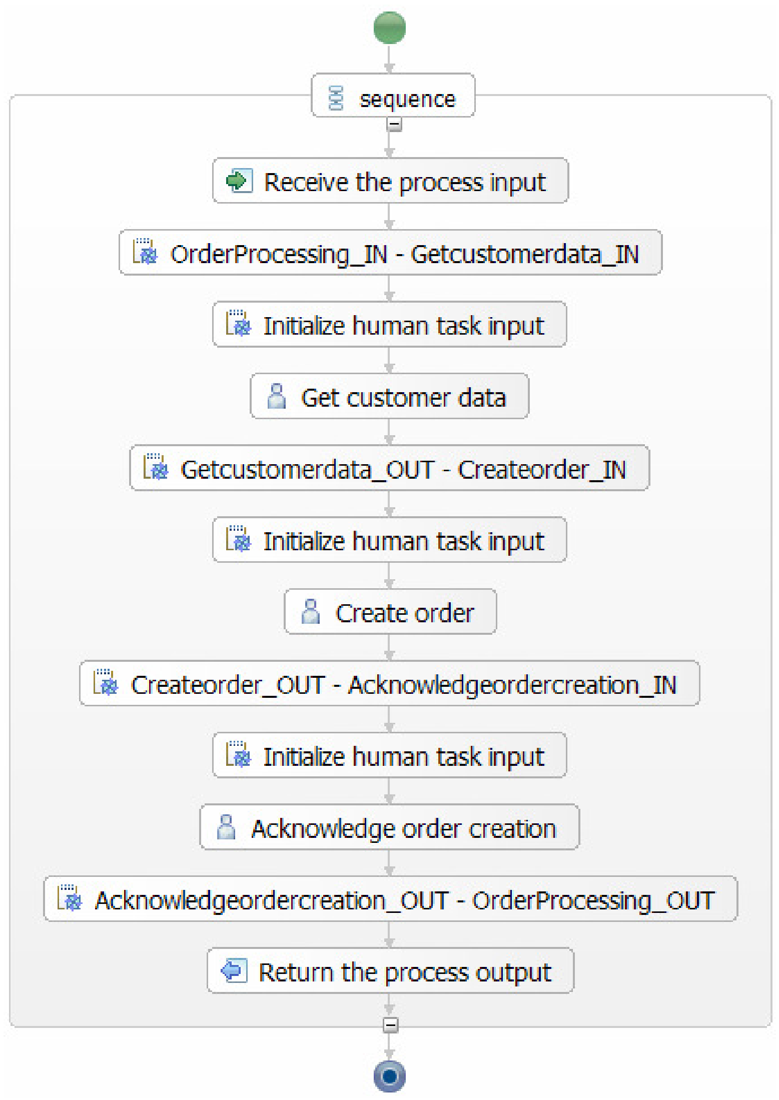

sequence
Receive the process input
OrderProcessing_IN - Getcustomerdata_IN
Initialize human task input
Get customer data
Getcustomerdata_OUT - Createorder_IN
Initialize human task input
Create order
Createorder_OUT - Acknowledgeordercreation_IN
Initialize human task input
Acknowledge order creation
Acknowledgeordercreation_OUT - OrderProcessing_OUT
Return the process output

Figure 88: BPEL activity chain without redundant structured activities

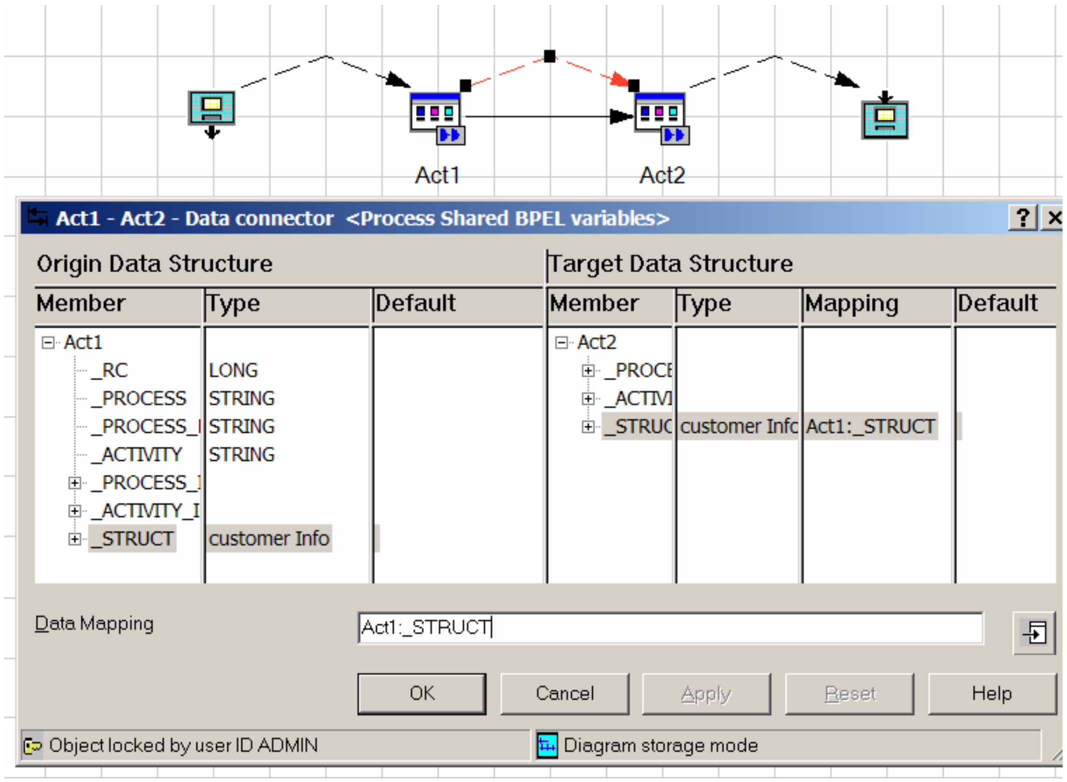
Removing redundant Java snippets and BPEL variables

The FDL2BPEL Conversion Tool sometimes generates redundant snippets and BPEL variables.

The FDL2BPEL Conversion Tool uses an extensive set of generic mapping rules that sometimes lead to suboptimal solutions. For instance, a data connector that passes the content of an activity output container "as is" to the input of another activity would be most easily mapped to a single BPEL variable that is shared by the two activities as the output variable and the input variable respectively. Instead of this you will find in the migrated BPEL model two generated BPEL variables and a snippet that copies the data from the output variable to the input variable.

In this case, you should manually modify the BPEL model in the WID business process editor, such that it uses the preferred "shared" BPEL variables.

FDL example:



Act1 - Act2 - Data connector <Process Shared BPEL variables>

Origin Data Structure

Member	Type	Default
⊟ Act1		
_RC	LONG	
_PROCESS	STRING	
_PROCESS_I	STRING	
_ACTIVITY	STRING	
⊞ _PROCESS_I		
⊞ _ACTIVITY_I		
⊞ _STRUCT	customer Info	

Target Data Structure

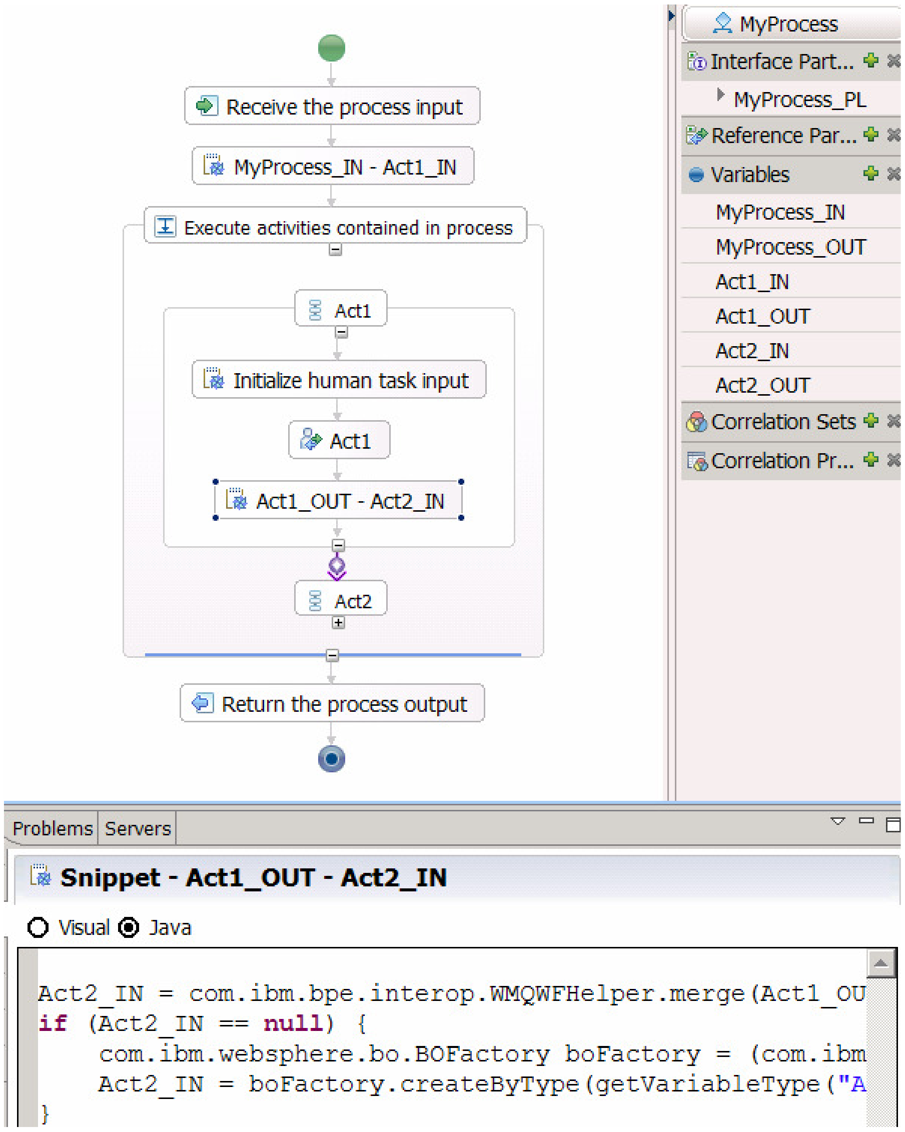
Member	Type	Mapping	Default
⊟ Act2			
⊞ _PROCE			
⊞ _ACTIVI			
⊞ _STRUC	customer Info	Act1:_STRUCT	

Data Mapping: Act1:_STRUCT

Object locked by user ID ADMIN | Diagram storage mode

Figure 89: FDL data connector with "_STRUCT to _STRUCT" mapping

Figure 90 shows how FDL2BPEL Conversion Tool translates the data connector to BPEL by inserting Java snippet "Act1_OUT – Act2_IN". The snippet is redundant, because it copies data without any change from BPEL variable "Act1_OUT" to variable "Act2_IN":



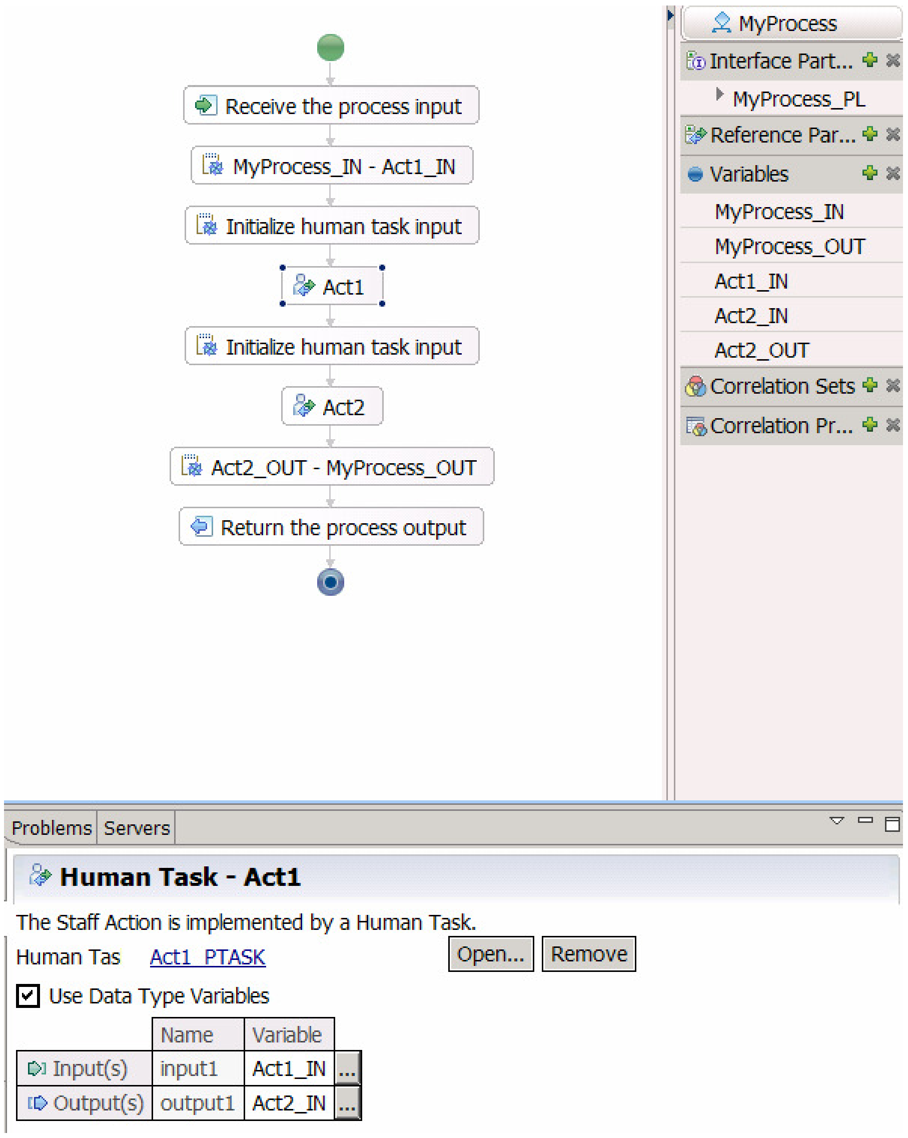
Visual [✓ Java]

```
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OU
if (Act2_IN == null) {
    com.ibm.websphere.bo.BOFactory boFactory = (com.ibm
    Act2_IN = boFactory.createByType(getVariableType("A
}
```

Figure 90: BPEL process with redundant Java snippet

You can improve the performance of the generated BPEL process, as follows (see Figure 91):

1. Remove the redundant structured activities.
2. Remove Java snippet "Act1_OUT – Act2_IN".
3. Remove BPEL variable "Act1_OUT".
4. Assign BPEL variable "Act2_IN" to the output of activity "Act1".



Human Task - Act1

The Staff Action is implemented by a Human Task.

Human Tas Act1_PTASK Open... Remove

☑ Use Data Type Variables

	Name	Variable
Input(s)	input1	Act1_IN
Output(s)	output1	Act2_IN

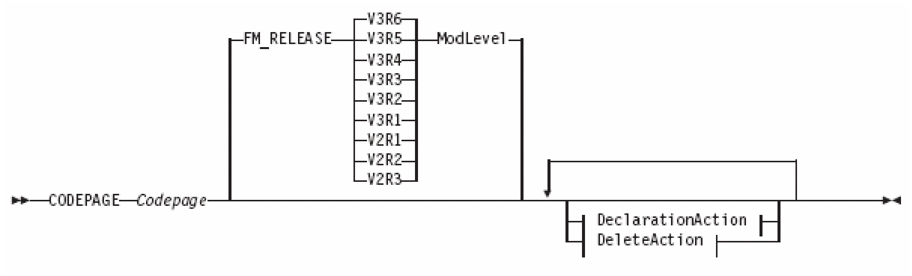
Figure 91: Improved BPEL process with shared variable

Appendix 1: Migration coverage listed by FDL syntax expressions

Introduction

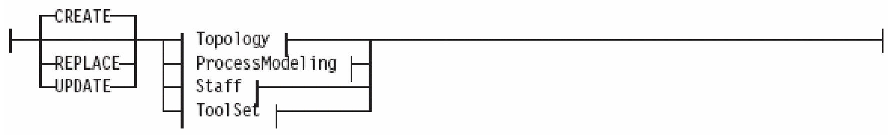
Not all constructs that you can specify in FDL can be mapped in a one-to-one fashion to a corresponding BPEL construct due to the differences in the programming models. Appropriate warnings in the output of the FDL2BPEL Conversion Tool are provided as XML comments in the generated artifacts (such as the files with the extension ".bpel"). These warnings inform you about constructs that are not migrated in a one-to-one fashion. This chapter and the following[64] help you to identify limitations that might impact your WMQWF source artifact migration.

FDL source file



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no work-around possible	See page
CODEPAGE Codepage		x[65]			
FM_RELEASE VnRn ModeLevel		x[66]			
DeclarationAction					104
DeleteAction			x[67]		

DeclarationAction



[64] See "Appendix 2: Migration hints" on page 130.

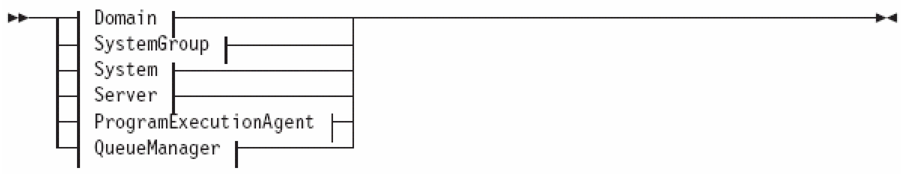
[65] See "FDL with codepage different from system codepage" on page 130.

[66] See "Migration of early FDL versions" on page 130.

[67] See "FDL processing actions" on page 130.

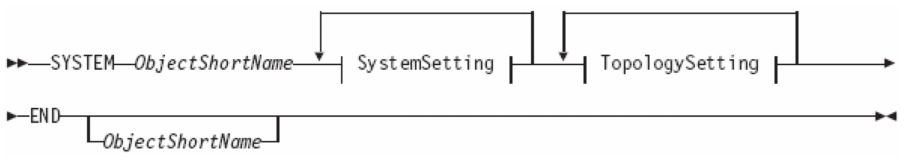
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
CREATE			x[68]		
REPLACE			x[68]		
UPDATE			x[68]		
Topology					105
ProcessModeling					111
Staff			x[69]		
ToolSet				x[70]	

Topology



FDL syntax expression[71]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
Domain			x		
SystemGroup			x		
System					105
Server					109
ProgramExecutionAgent				x	
QueueManager			x		

System



[68] See "FDL processing actions" on page 130.

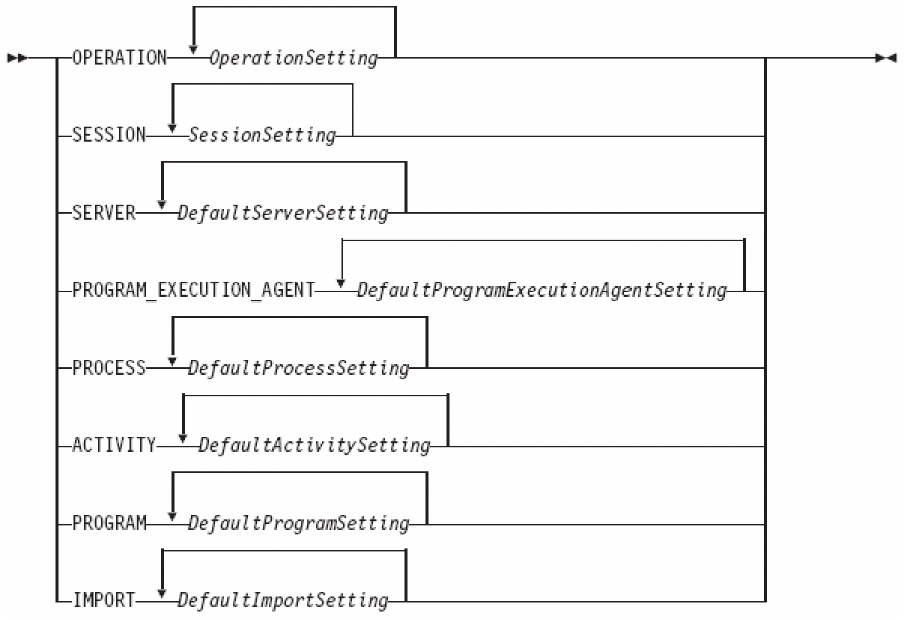
[69] See "WMQWF staff repository" on page 137.

[70] See "Migration of WMQWF Buildtime tool set definitions not supported" on page 131.

[71] See "System network" on page 141.

FDL syntax expression[72]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
SYSTEM ObjectShortName			x		
SystemSetting			x		
TopologySetting					106

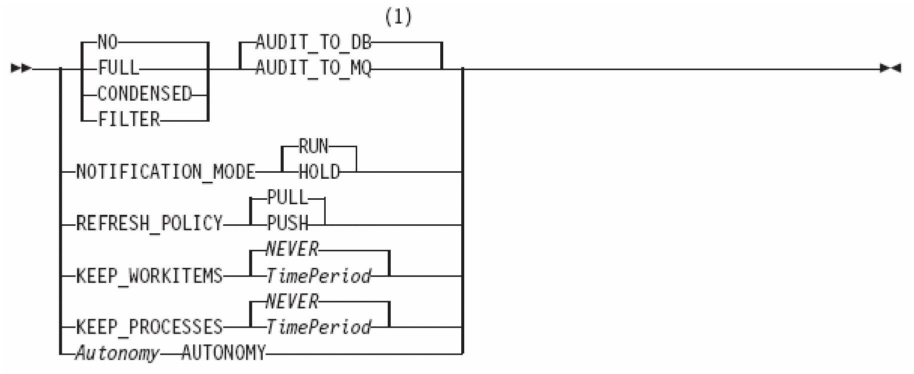
TopologySetting



[72] See "System network" on page 141.

FDL syntax expression[73]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
OPERATION *OperationSetting*			x		
SESSION *SessionSetting*			x		
SERVER *DefaultServerSetting*			x		
PROGRAM_EXECUTION_AGENT *DefaultProgramExecutionAgentSetting*			x		
PROCESS ***DefaultProcessSetting***					107
ACTIVITY ***DefaultActivitySetting***					109
PROGRAM *DefaultProgramSetting*			x		
IMPORT *DefaultImportSetting*			x		

DefaultProcessSetting



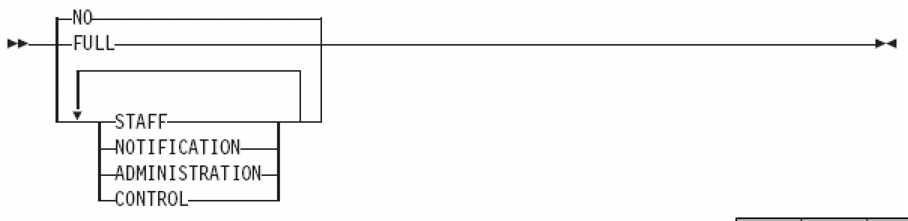
Notes:

1 Instead of specifying AUDIT_TO_DB or AUDIT_TO_MQ, you can use AUDIT, which was a valid option in previous versions of MQ Workflow. With the AUDIT option, you define AUDIT_TO_DB. For details, refer to the *IBM WebSphere MQ Workflow: Administration Guide.*

[73] See "Properties inherited from domain or system" on page 141.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no work-around possible	See page
NO (AUDIT_TO_DB | AUDIT_TO_MQ | AUDIT)	x				
(FULL | CONDENSED | FILTER) (AUDIT_TO_DB | AUDIT_TO_MQ | AUDIT)		x[74]			
NOTIFICATION_MODE RUN	x				
NOTIFICATION_MODE HOLD				x[75]	
REFRESH_POLICY PULL	x				
REFRESH_POLICY PUSH				x[76]	
KEEP_WORKITEMS (NEVER | ***TimePeriod***)				x[77]	
KEEP_PROCESSES (NEVER | ***TimePeriod***)	x[78]				
TimePeriod					129
Autonomy AUTONOMY					108

Autonomy



FDL syntax expression[79]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no work-around possible	See page
NO	x				
FULL				x	
STAFF				x	
NOTIFICATION				x	
ADMINISTRATION				x	
CONTROL	x				

[74] See "Audit settings" on page 141.

[75] See "Notification mode" on page 137.

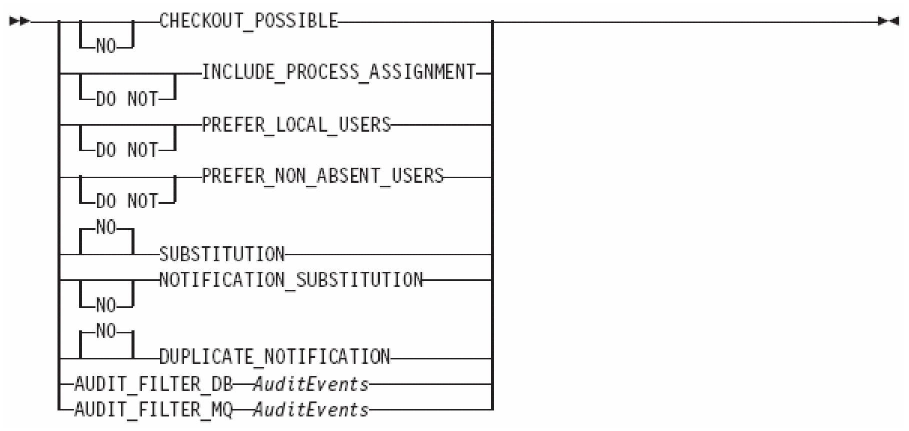
[76] See "Workitem refresh policy" on page 144.

[77] See "Policy for how to deal with finished workitems" on page 144.

[78] See "Policy for how to deal with finished process instances" on page 144.

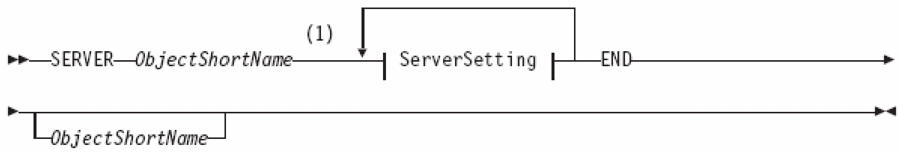
[79] See "Process autonomy modes" on page 141.

DefaultActivitySetting



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no work-around possible	See page
(NO) CHECKOUT_POSSIBLE	x[80]				
INCLUDE_PROCESS_ASSIGNMENT				x[81]	
DO NOT INCLUDE_PROCESS_ASSIGNMENT	x				
PREFER_LOCAL_USERS				x[81]	
DO NOT PREFER_LOCAL_USERS	x				
(DO NOT) PREFER_NON_ABSENT_USERS	x				
(NO) SUBSTITUTION	x				
NOTIFICATION_SUBSTITUTION				x[82]	
NO NOTIFICATION_SUBSTITUTION	x				
DUPLICATE_NOTIFICATION				x[82]	
NO DUPLICATE_NOTIFICATION	x				
AUDIT_FILTER_DB *AuditEvents*		x[83]			
AUDIT_FILTER_MQ *AuditEvents*		x[83]			

Server



[80] Serves as indicator for a "staff activity" (See "Activity classification rules" on page 60).

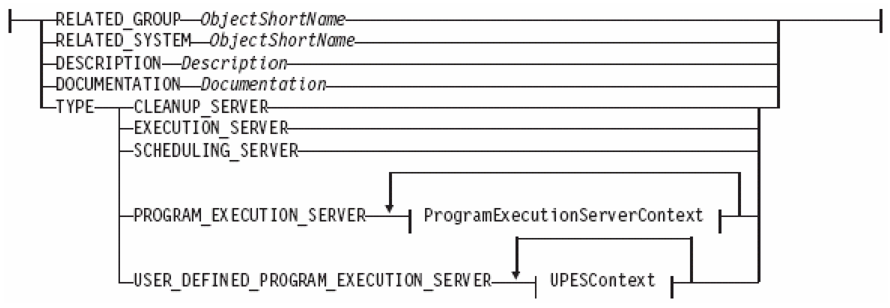
[81] See "Staff assignment criteria" on page 139.

[82] See "Notification policy" on page 138.

[83] See "Audit settings" on page 141.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
SERVER *ObjectShortName*	x[84]				
ServerSetting					110

ServerSetting

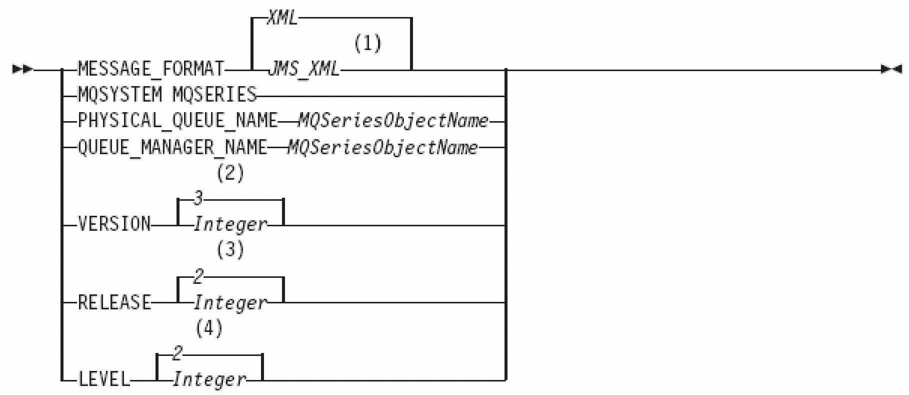


RELATED_GROUP ObjectShortName
RELATED_SYSTEM ObjectShortName
DESCRIPTION Description
DOCUMENTATION Documentation
TYPE
CLEANUP_SERVER
EXECUTION_SERVER
SCHEDULING_SERVER
PROGRAM_EXECUTION_SERVER
ProgramExecutionServerContext
USER_DEFINED_PROGRAM_EXECUTION_SERVER
UPESContext

FDL syntax expression[84]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
RELATED_GROUP *ObjectShortName*	x				
RELATED_SYSTEM *ObjectShortName*	x				
DESCRIPTION *Description*				x	
DOCUMENTATION *Documentation*				x	
TYPE CLEANUP_SERVER				x	
TYPE EXECUTION_SERVER				x	
TYPE SCHEDULING_SERVER				x	
TYPE PROGRAM_EXECUTION_SERVER ProgramExecutionServerContext				x	
TYPE USER_DEFINED_PROGRAM_EXECUTION_SERVER ***UPESContext***					111

[84] See "User-defined program execution server (UPES)" on page 139.

UPESContext



Notes:

1 This means that you use a JMS-compliant XML format.

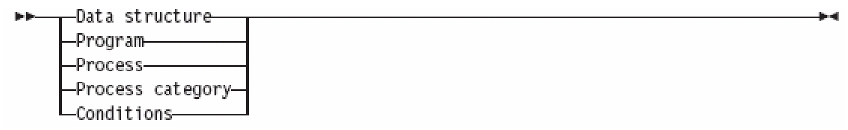
2 Default is set only if the FDL file contains a version number prior to Version 3 Release 3.

3 Default is set only if the FDL file contains a release number prior to Version 3 Release 3.

4 Default is set only if the FDL file contains a level number prior to Version 3 Release 3.

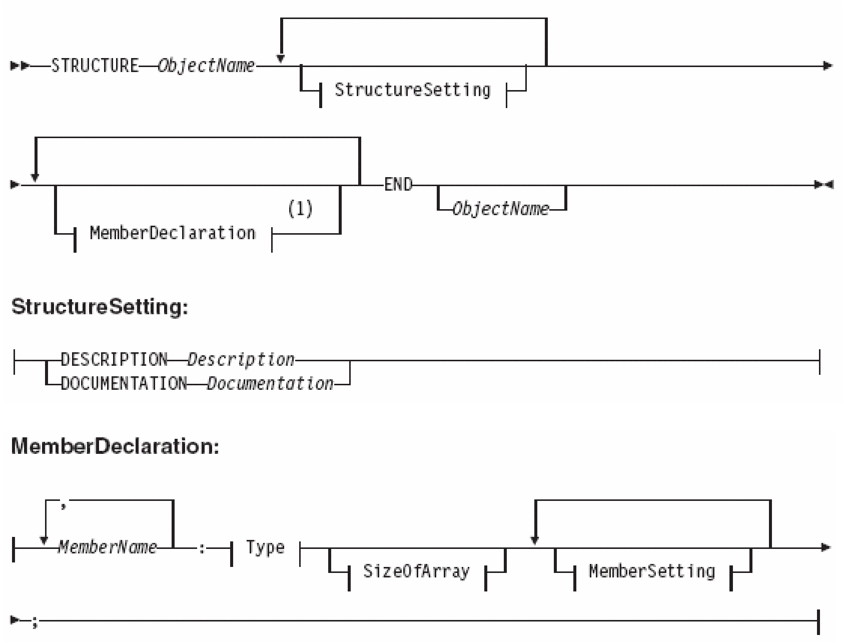
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
MESSAGE_FORMAT *XML*	x				
MESSAGE_FORMAT *JMS_XML*	x				
MQSYSTEM MQSERIES				x	
PHYSICAL_QUEUE_NAME *MQSeriesObjectName*	x				
QUEUE_MANAGER_NAME *MQSeriesObjectName*	x				
VERSION (3 | *Integer*)				x	
RELEASE (2 | *Integer*)				x	
LEVEL (2 | *Integer*)				x	

ProcessModeling



FDL syntax expression[85] [86]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
Data structure					112
Program					113
Process					114
Process category					127
Conditions					128

Data structure

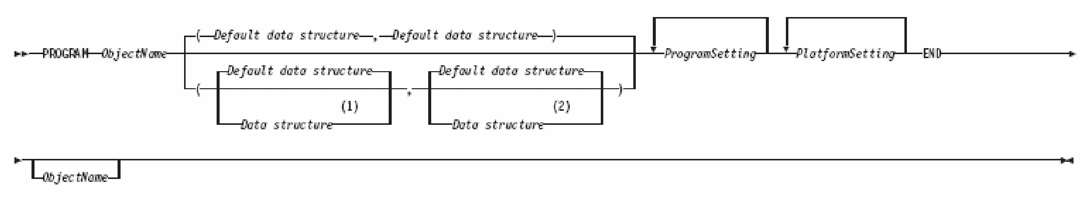


85 See "User-defined program execution server (UPES)" on page 139.

86 See "XML message types" on page 140.

FDL syntax expression[87]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
ObjectName	x				
StructureSetting		x[87]			
MemberDeclaration				x[88]	

Program



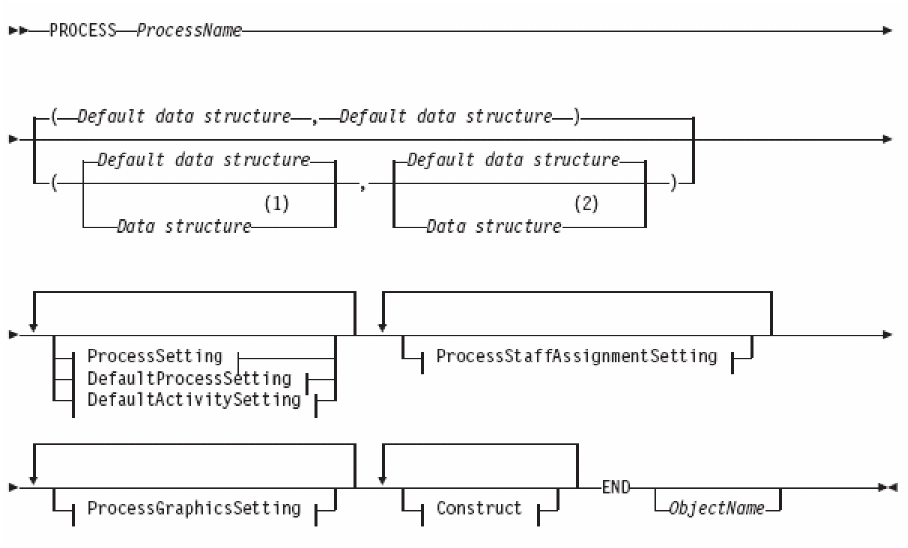
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROGRAM *ObjectName*	x				
Default data structure	x				
Data structure					112
ProgramSetting				x[89]	
PlatformSetting				x[89]	

[87] See "Data description and documentation" on page 140.

[88] See "Data member description and documentation" on page 140.

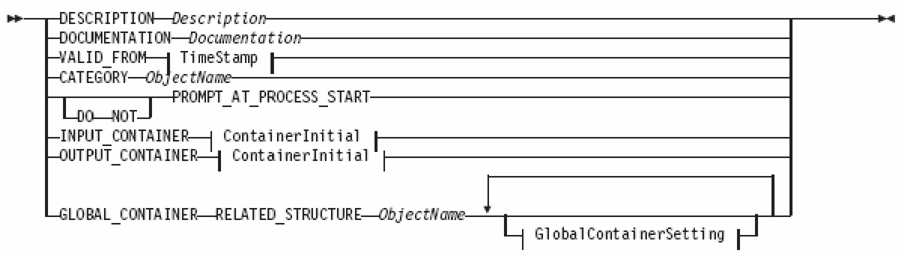
[89] See "Program execution properties" on page 142.

Process



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS *ProcessName*	x				
Default data structure	x				
Data structure					112
ProcessSetting					114
DefaultProcessSetting					107
DefaultActivitySetting					109
ProcessStaffAssignmentSetting					115
ProcessGraphicSetting		x[90]			
Construct					116

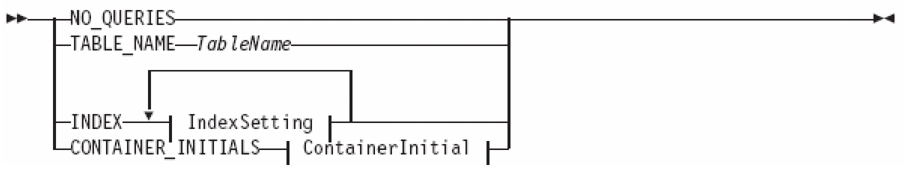
ProcessSetting



[90] See "Graphical layout information" on page 131.

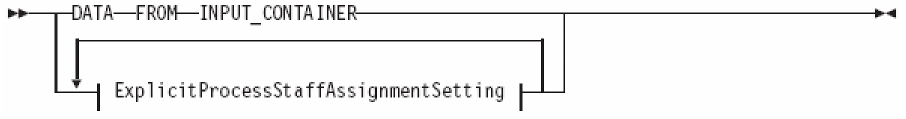
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
DESCRIPTION *Description*	x				
DOCUMENTATION *Documentation*	x				
VALID FROM TimeStamp	x				
CATEGORY *ObjectName*	x[91]				
PROMPT_AT_PROCESS_START	x				
DO NOT PROMPT_AT_PROCESS_START				x[92]	
INPUT_CONTAINER ContainerInitial				x[93]	
OUTPUT_CONTAINER ContainerInitial	x				
GLOBAL_CONTAINER RELATED_STRUCTURE *ObjectName*	x				
GlobalContainerSetting					115

GlobalContainerSetting



FDL syntax expression[94]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
NO_QUERIES				x	
TABLE_NAME *TableName*				x	
INDEX IndexSetting				x	
CONTAINER_INITIALS ContainerInitial	x				

ProcessStaffAssignmentSetting



[91] See "Staff assignment by process category" on page 138.

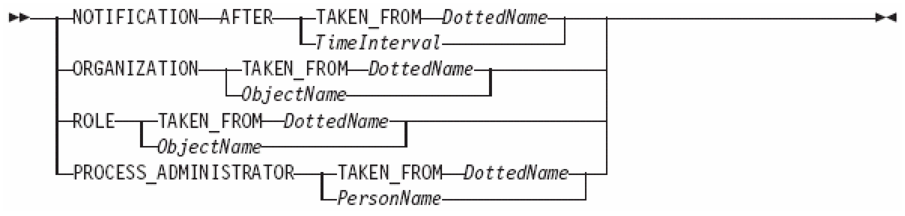
[92] See "Prompting for data at process start" on page 133.

[93] See "Initial process input data" on page 133.

[94] See "Global container database settings" on page 134.

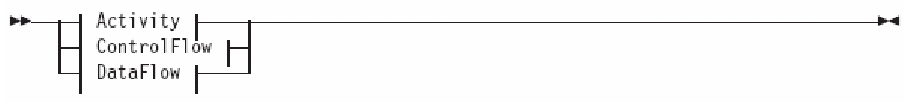
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
DATA FROM INPUT_CONTAINER				x[95]	
ExplicitProcessStaffAssignmentSetting					116

ExplicitProcessStaffAssignmentSetting



FDL syntax expression[96]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
NOTIFICATION AFTER TAKEN_FROM *DottedName*				x	
NOTIFICATION AFTER *TimeInterval*				x	
ORGANIZATION TAKEN_FROM *DottedName*	x				
ORGANIZATION *ObjectName*	x				
ROLE TAKEN_FROM *DottedName*	x				
ROLE *ObjectName*	x				
PROCESS_ADMINISTRATOR TAKEN_FROM *DottedName*	x				
PROCESS_ADMINISTRATOR *PersonName*	x				

Construct

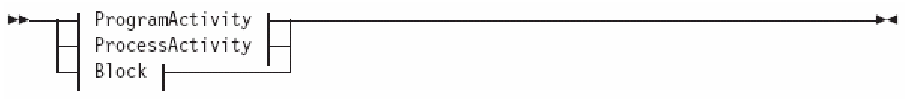


[95] See "Staff assignment from input container" on page 138.

[96] See "Process notification" on page 138.

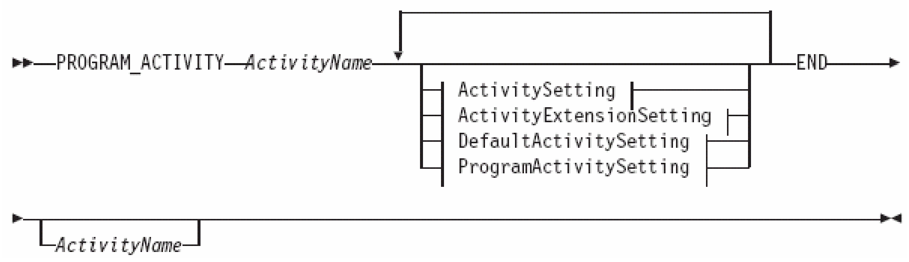
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
Activity					117
ControlFlow					126
DataFlow					126

Activity



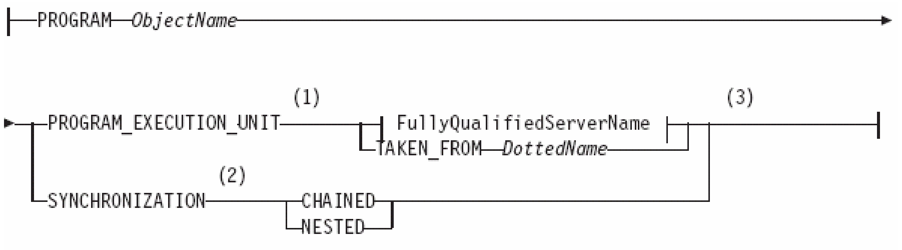
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
ProgramActivity					117
ProcessActivity					119
Block					120

ProgramActivity



FDL syntax expression[97]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROGRAM_ACTIVITY *ActivityName*	x				
ActivitySetting					121
ActivityExtensionSetting					122
DefaultActivitySetting					109
ProgramActivitySetting					118

ProgramActivitySetting



Notes:

1 The keyword PROGRAM_EXECUTION_SERVER can still be used instead of PROGRAM_EXECUTION_UNIT. However, for new definitions, use only PROGRAM_EXECUTION_UNIT, because the old keyword is only valid as an interim solution for this release. The type of server can be a PROGRAM_EXECUTION_SERVER or a USER_DEFINED_PROGRAM_EXECUTION_SERVER.

2 If **Asynchronous** was specified on the Execution page of the **Program activity properties**, this appears as CHAINED in the FDL file. If **Synchronous** was specified on the Execution page of the **Program activity properties**, this appears as NESTED in the FDL file.

3 If the following conditions apply, the activity is an empty activity:

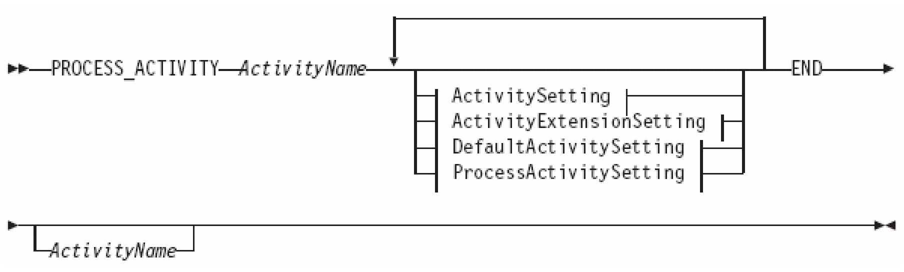
- SYNCHRONIZATION CHAINED is specified. In the Buildtime user interface, this is **Asynchronous Mode**.
- PROGRAM FMCINTERNALNOOP is specified and this program is defined.
- The activity is started automatically.
- Input and output data structures are the same.

During run time, if an empty activity is started, a program is not called to execute the activity and it is immediately completed. If a data default connector is defined for the activity, the specified mappings are executed from the activity input container to the activity output container.

[97] See "FDL ProgramActivity" on page 142.

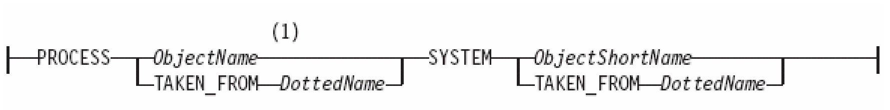
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROGRAM *ObjectName*	x				
PROGRAM_EXECUTION_UNIT *FullyQualifiedServerName*	x				
PROGRAM_EXECUTION_UNIT TAKEN_FROM *DottedName*				x[98]	
SYNCHRONIZATION CHAINED	x				
SYNCHRONIZATION NESTED	x				

ProcessActivity



FDL syntax expression[99]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS_ACTIVITY ActivityName	x				
ActivitySetting					121
ActivityExtensionSetting					122
DefaultActivitySetting					109
ProcessActivitySetting					119

ProcessActivitySetting



Notes:

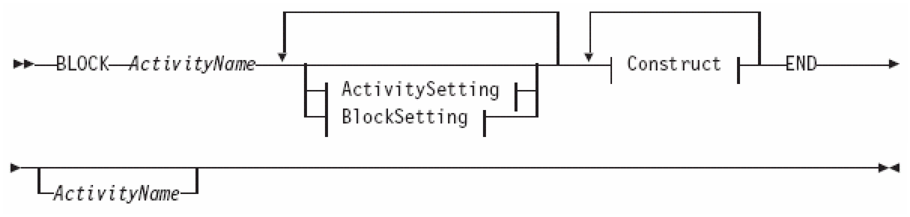
1 This is the name of a process.

[98] See "Program execution unit taken from container" on page 140.

[99] See "Staff assignment and notification criteria assigned to process activities" on page 139.

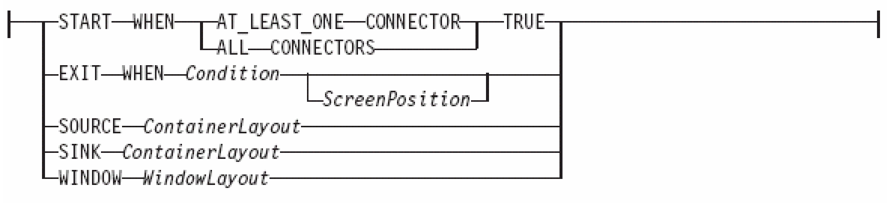
FDL syntax expression[100]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS ObjectName	x				
PROCESS TAKEN_FROM *DottedName*	x				
SYSTEM *ObjectShortName*				x	
SYSTEM TAKEN_FROM *DottedName*				x	

Block



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
BLOCK *ActivityName*	x				
ActivitySetting					121
BlockSetting					120
Construct					116

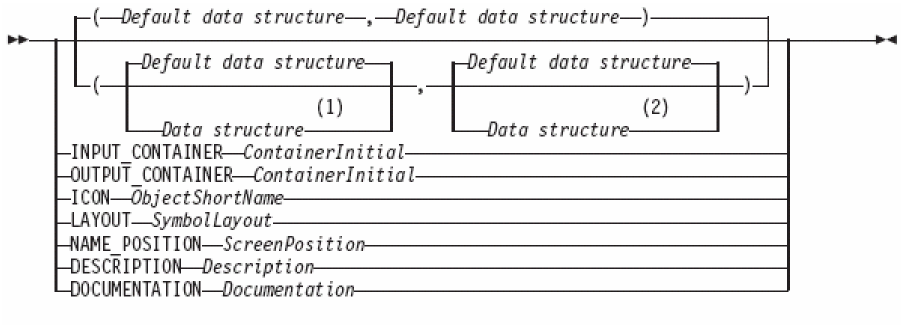
BlockSetting



[100] See "System properties assigned to process activities" on page 142.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
START WHEN AT_LEAST_ONE CNNECTOR TRUE	x				
START WHEN ALL CNNECTORS TRUE	x				
EXIT WHEN ***Condition***					128
ScreenPosition		x[101]			
SOURCE	x				
SINK	x				
ContainerLayout		x[101]			
WINDOW *WindowLayout*		x[101]			

ActivitySetting


(—Default data structure—,—Default data structure—)
(—Default data structure (1) Data structure—,—Default data structure (2) Data structure—)
INPUT_CONTAINER—ContainerInitial
OUTPUT_CONTAINER—ContainerInitial
ICON—ObjectShortName
LAYOUT—SymbolLayout
NAME_POSITION—ScreenPosition
DESCRIPTION—Description
DOCUMENTATION—Documentation

Notes:

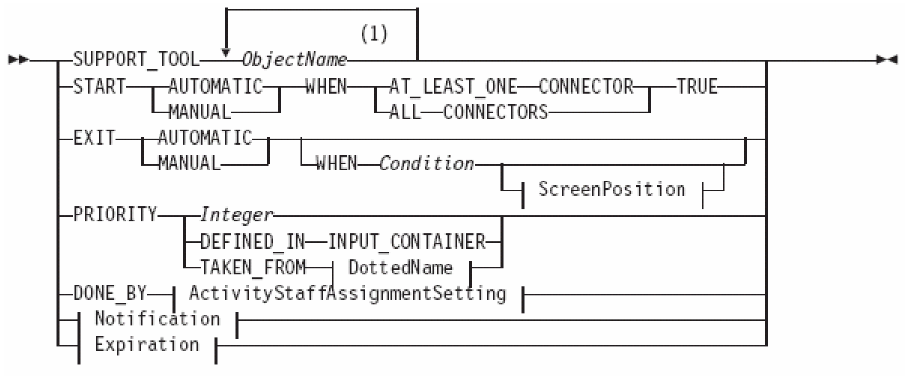
1 The first data structure that you specify is the input data structure.

2 The second data structure that you specify is the output data structure.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
Default data structure	x				
Data structure	x				
INPUT_CONTAINER *ContainerInitial*	x				
OUTPUT_CONTAINER *ContainerInitial*	x				
ICON *ObjectShortName*				x[101]	
LAYOUT *SymbolLayout*		x[101]			
NAME_POSITION *ScreenPosition*				x[101]	
DESCRIPTION *Description*	x				
DOCUMENTATION *Documentation*	x				

[101] See "Graphical layout information" on page 131.

ActivityExtensionSetting

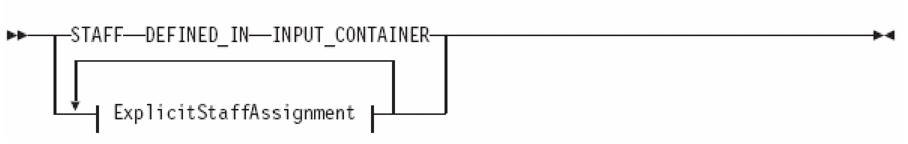


Notes:

1 This is the name of a program.

FDL syntax expression	**Migration supported with FDL2BPEL Conversion**	**Workaround possible with manual rework**	**Outside the scope of process model migration**	**Not applicable, no workaround possible**	**See page**
SUPPORT_TOOL ObjectName				x[102]	
START AUTOMATIC	x[103]				
START MANUAL		x[103]			
START ... WHEN AT_LEAST_ONE CONNECTOR TRUE	x				
START ... WHEN ALL CONNECTORS TRUE	x				
EXIT AUTOMATIC	x[103]				
EXIT MANUAL		x[103]			
EXIT ... WHEN ***Condition***	x				
Condition					128
ScreenPosition				x[104]	
PRIORITY *Integer*	x				
PRIORITY DEFINED_IN INPUT_CONTAINER	x				
PRIORITY TAKEN_FROM DottedName	x				
DONE_BY ***ActivityStaffAssignmentSetting***					122
Notification					124
Expiration					125

ActivityStaffAssignmentSetting



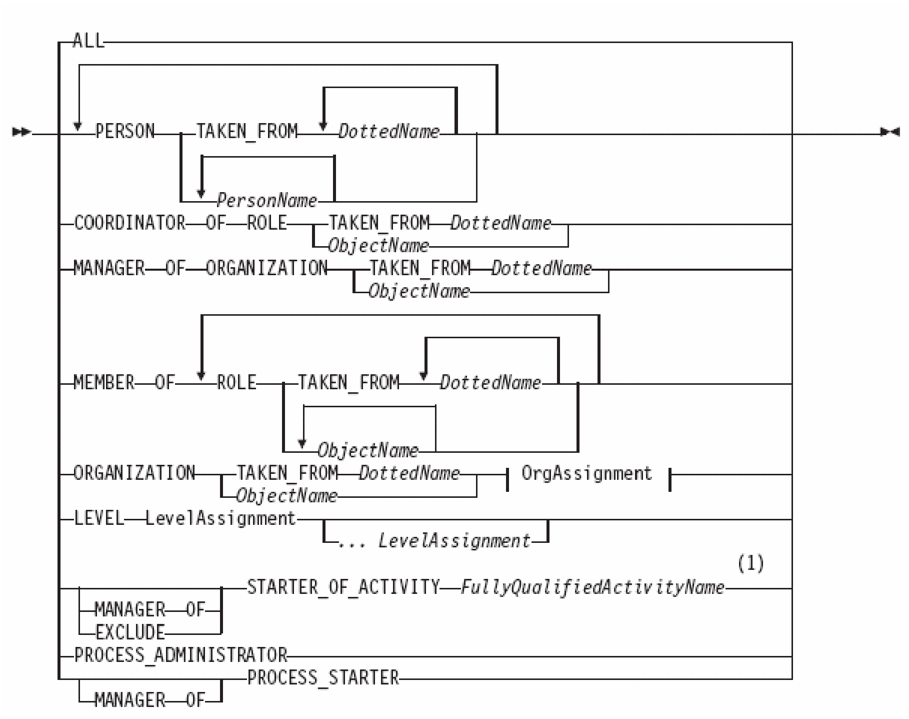
[102] See "Support tools" on page 143.

[103] See "Activity start and exit mode" on page 132.

[104] See "Graphical layout information" on page 131.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
STAFF DEFINED_IN INPUT_CONTAINER				x[105]	
ExplicitStaffAssignment					123

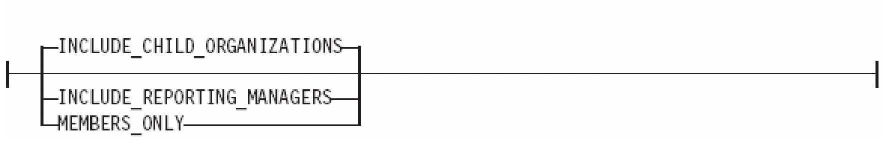
ExplicitStaffAssignment



[105] See "Staff assignment from input container" on page 138.

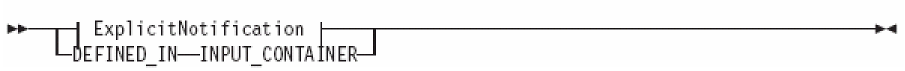
FDL syntax expression[106]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
ALL	x				
PERSON TAKEN_FROM *DottedName*	x				
PERSON PersonName	x				
COORDINATOR OF ROLE TAKEN_FROM *DottedName*				x	
COORDINATOR OF ROLE *ObjectName*				x	
MANAGER OF ORGANIZATION TAKEN_FROM *DottedName*				x	
MANAGER OF ORGANIZATION *ObjectName*				x	
MEMBER OF ROLE TAKEN_FROM *DottedName*	x				
MEMBER OF ROLE *ObjectName*	x				
ORGANIZATION TAKEN_FROM	x				
ORGANIZATION *ObjectName*	x				
OrgAssignment					124
LEVEL LevelAssignment				x	
MANAGER OF STARTER_OF_ACTIVITY *FullyQualifiedActivityName*	x				
EXCLUDE STARTER_OF_ACTIVITY *FullyQualifiedActivityName*		x[106]			
PROCESS_ADMINISTRATOR	x				
PROCESS_STARTER	x				
MANAGER OF PROCESS_STARTER	x				

OrgAssignment



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
INCLUDE_CHILD_ORGANIZATIONS	x				
INCLUDE_REPORTING_MANAGERS		x[106]			
MEMBERS_ONLY	x				

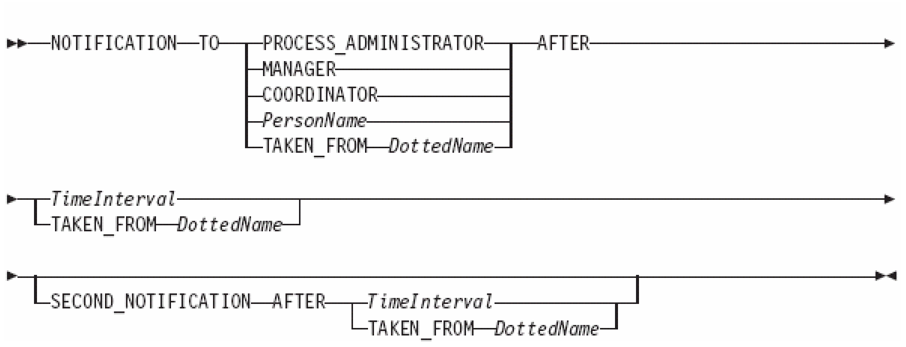
Notification



[106] See "Staff assignment criteria" on page 139.

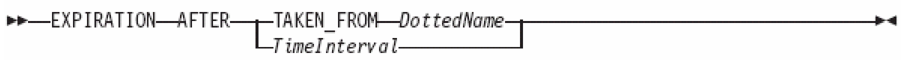
FDL syntax expression[107]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
ExplicitNotification					125
DEFINED_IN INPUT_CONTAINER				x[108]	

ExplicitNotification



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
NOTIFICATION TO PROCESS_ADMINISTRATOR	x				
NOTIFICATION TO MANAGER	x				
NOTIFICATION TO COORDINATOR		x[109]			
NOTIFICATION TO *PersonName*	x				
NOTIFICATION TO TAKEN_FROM	x				
NOTIFICATION … AFTER *TimeInterval*	x				
NOTIFICATION … AFTER TAKEN_FROM *DottedName*				x[108]	
SECOND_NOTIFICATION AFTER *TimeInterval*	x				
SECOND_NOTIFICATION AFTER TAKEN_FROM *DottedName*				x[108]	

Expiration



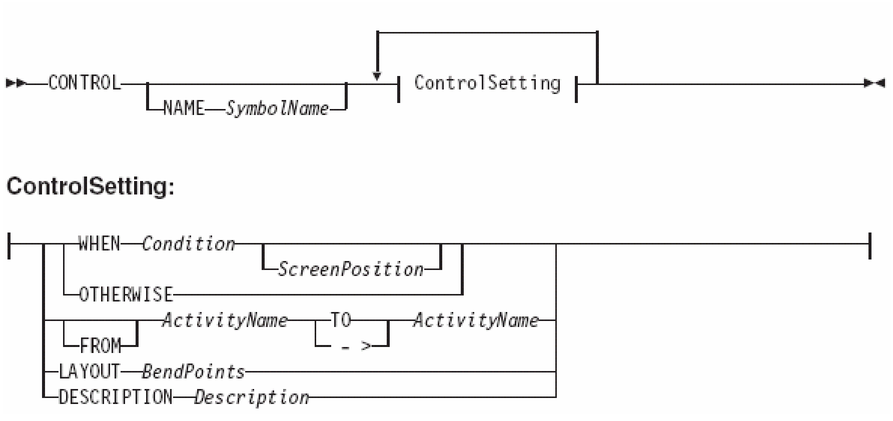
[107] See "Notification" on page 137.

[108] See "Notification from container" on page 138.

[109] See "Staff assignment criteria" on page 139.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
EXPIRATION AFTER TAKEN_FROM *DottedName*	x				
EXPIRATION AFTER *TimeInterval*	x				

ControlFlow

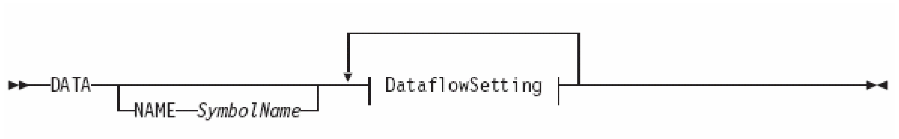


FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
CONTROL	x				
NAME *SymbolName*				x[110]	
WHEN ***Condition***					128
OTHERWISE	x				
FROM? *ActivityName* (TO | ->) *ActivityName*	x				
LAYOUT *BendPoints*				x[111]	
DESCRIPTION *Description*	x				

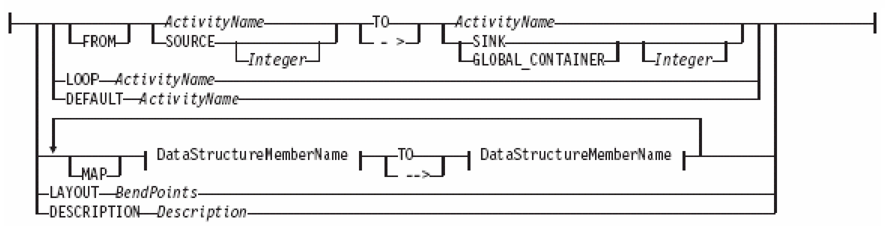
DataFlow

[110] See "Data and control connector" on page 132.

[111] See "Graphical layout information" on page 131.

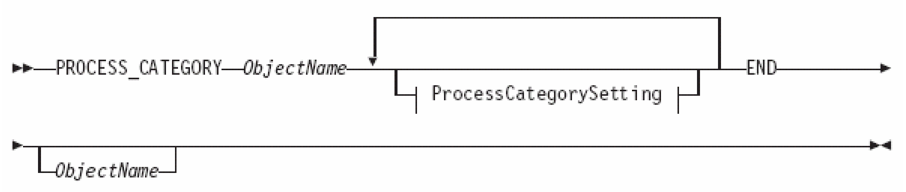


DataflowSetting:

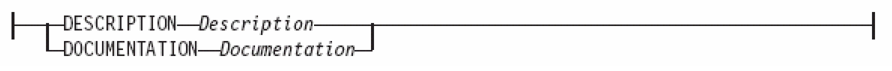


FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
DATA	x				
NAME *SymbolName*				x[112]	
FROM? (*ActivityName* | SOURCE *Integer*?) (TO | ->) (*ActivityName* | SINK *Integer*? | GLOBAL_CONTAINER *Integer*?)	x				
LOOP *ActivityName*	x				
DEFAULT *ActivityName*	x				
MAP? DataStructureMemberName (TO | ->) DataStructureMemberName	x				
LAYOUT *BendPoints*				x[113]	
DESCRIPTION *Description*	x				

Process category



ProcessCategorySetting:



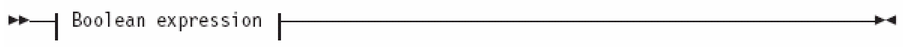
[112] See "Data and control connector" on page 132.

[113] See "Graphical layout information" on page 131.

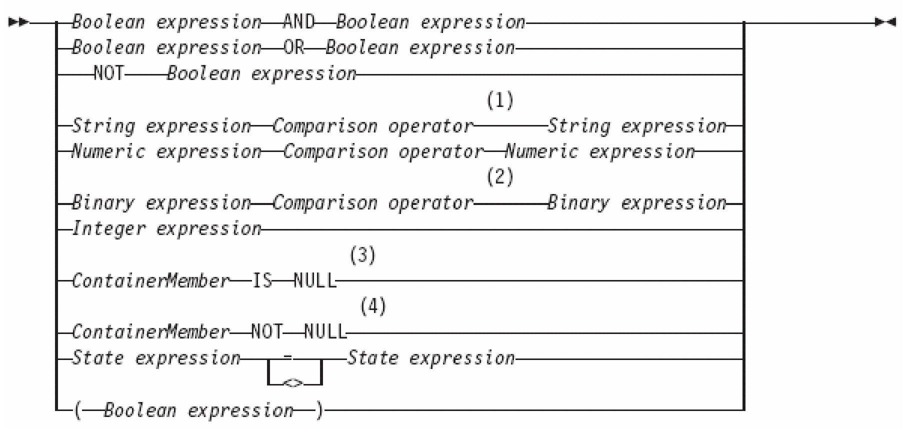
FDL syntax expression[114]	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS_CATEGORY *ObjectName*	x[115]				
DESCRIPTION *Description*				x	
DOCUMENTATION *Description*				x	

Conditions

Condition



Boolean expression


Boolean expression—AND—Boolean expression
Boolean expression—OR—Boolean expression
NOT—Boolean expression
(1)
String expression—Comparison operator—String expression
Numeric expression—Comparison operator—Numeric expression
(2)
Binary expression—Comparison operator—Binary expression
Integer expression
(3)
ContainerMember—IS—NULL
(4)
ContainerMember—NOT—NULL
State expression
=
<>
State expression
(—Boolean expression—)

Notes:

1 Strings are compared character by character based on the value of their ASCII character codes.

2 The valid comparison operators are = and <> only.

3 Use this operator for querying whether a container member is not set.

4 Use this operator for querying whether a container member is set.

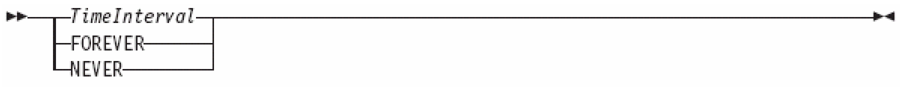
[114] See "Process category description and documentation" on page 141.

[115] See "Staff assignment by process category" on page 138.

FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
Boolean expression (AND | OR) *Boolean expression*	x				
NOT *Boolean expression*	x				
String expression Comparison operator String expression	x				
Numeric expression Comparison operator Numeric expression	x				
Binary expression Comparison operator Binary expression	x				
Integer expression	x				
Container member IS NULL	x				
Container member NOT NULL	x				
State expression (= | <>) *State expression*	x[116]				

Common variables

TimePeriod



FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
TimeInterval				x[117]	
FOREVER	x				
NEVER	x				

[116] See "Activity states" on page 133.

[117] See "Policy for how to deal with finished process instances" on page 144.

Appendix 2: Migration hints

Due to differences in technology and programming model you cannot always expect a seamless migration of your WMQWF processes to Business Process Choreographer. The following sections help you to learn about known migration limitations and what actions you can take to overcome them.

FDL source file

FDL with codepage different from system codepage

Description

It is not possible to migrate an FDL file with another codepage than the system codepage.[118]

Hint

Take care that the computer where you have created the FDL file (exported from WMQWF Buildtime) has the same installed codepage as the computer were you run the migration. You could also convert the FDL file to the required system codepage. For example, this can be done with some text editors where you can specify the codepage when reading and writing files.

Migration of early FDL versions

Description

Migration can fail with FDL input files containing a version ID prior to "FM_RELEASE V3R6".[118]

Explanation

The FDL2BPEL Conversion Tool is based on version "V3R6". In particular, the "FM_RELEASE" keyword that you find in the FDL source code is ignored. Older FDL files might contain deprecated syntactical scripts that are not expected by the initial FDL parsing step.

Hint

It is still worth making a trial migration with an older FDL file, because not every FDL file will be subject to version dependent differences in the syntax. But, if the FDL2BPEL Conversion Tool fails, you need to run a version upgrade of your FDL file using WMQWF Buildtime with V3.6 or later before you run the FDL to BPEL migration again.

FDL processing actions

Description

The FDL2BPEL conversion tool treats the actions "CREATE", "REPLACE", and "UPDATE" the same way and expects the complete FDL definition of the respective modeling object whenever it encounters one of these three keywords. FDL definitions preceded by “DELETE” are ignored. [118] [119]

Explanation

Declaration and delete actions are used as instructions on how to deal with the model constructs encountered in the FDL file that you import to the WMQWF runtime database. For instance, you might want to replace previously stored "person" definitions. Note that the FDL2BPEL Conversion Tool is not aware of the database system that will store the final BPEL

[118] See "FDL source file" on page 104.

[119] See "DeclarationAction" on page 104.

process models. That is why it handles the declaration and delete actions as explained in the following hint:

<u>**Hint**</u>

1. FDL constructs with or without declaration actions ("CREATE", "REPLACE", and "UPDATE") are treated the same way and migrated according to the documented mapping rules. That is why you should take care that FDL definitions preceded by "REPLACE" or "UPDATE" are self-contained and complete.
2. FDL constructs with preceding "DELETE" action are ignored and not migrated.

Problem detection

<u>**Description**</u>

The FDL2BPEL Conversion Tool expects a syntactical and semantically correct input file. The error reporting capabilities of this tool are restricted. Despite that the tool performs some simple consistency checks and makes you aware if indispensable FDL definitions are missing (for example, required subprocesses, data structures, programs, and UPES server definitions). Migration problems are reported as error and warning messages that may not always give you enough context information to locate the source of the problem easily (for example "Illegal data path expression").

<u>**Explanation**</u>

While the FDL2BPEL Conversion Tool generates the target constructs the context of a problem situation is sometimes unknown, such that the appropriate information is not available to give a helpful explanation. But the error message is always inserted as a comment annotation at the appropriate location in the generated (BPEL, WSDL, and so on) file where the problem occurred.

<u>**Hint**</u>

In such cases, it is helpful to locate the same message in the generated files using a source editor (usually the files with extension "*.bpel"). For instance, the above message "Illegal data path expression" may be found in a condition expression which refers to an undefined data member in the input variable of an activity.

In case of unexpected FDL parsing error messages which are not clear, import and translate the FDL to a WMQWF Runtime database. The FDL import tool "fmcibie" will check the FDL definitions and report errors, warnings, and informational messages on a much more detailed level.

Graphical business process editor

Migration of WMQWF Buildtime tool set definitions not supported

<u>**Description**</u>

You cannot migrate a customized WMQWF Buildtime tool set.[120]

<u>**Explanation**</u>

With WMQWF Buildtime, you can have a modeling tool bar with customized icons. There is no equivalent feature available in WebSphere Integration Developer (WID).

Graphical layout information

<u>**Description**</u>

The FDL2BPEL Conversion Tool does not migrate any graphical layout properties of a WMQWF business process.[121] [122] [123] [124] [125]

[120] See "DeclarationAction" on page 104.

[121] See "Process" on page 114.

Explanation

Layout information describes, for instance, the position of an activity node in the diagram view of WMQWF Buildtime. The graphical business process editor of WID has an "auto-layout" function for the automatic alignment of the nodes of "parallel activities". BPEL processes migrated from WMQWF will therefore look similar but not exactly the same as in the diagram view of WMQWF Buildtime.

Activity icons

Description

The FDL2BPEL Conversion Tool does not migrate user-defined icons assigned to activity nodes.[126]

Data and control connector names

Description

Names assigned to data or control connectors of a WMQWF process model cannot be migrated.[127] [128]

Control flow

Activity start and exit mode

Description

The FDL2BPEL Conversion Tool ignores the FDL activity start and exit modes "manual" and "automatic".[129]

Explanation

The start and exit modes are ignored, except for the "activity classification rules" which require the start mode to be "manual" in the case of a "staff activity".[130]

Hint

This means that FDL program activities with manual start mode and selected option "Can be checked out" will smoothly map to "human task" activities in BPEL. On the other hand, bear in mind that, if an FDL program activity has an automatic start mode, WMQWF runtime randomly selects a user as "activity starter" from the set of staff members resulting from staff resolution. Note that there is no equivalent behavior in WPS. Except for "human task" activities, all other BPEL activities start and finish automatically. In order to have a similar behavior like a manual activity start/exit you might insert "inline human tasks" that precede/follow the respective BPEL "Invoke" activity.[131]

---

[122] See "BlockSetting" on page 120.

[123] See "ActivitySetting" on page 121.

[124] See "ControlFlow" on page 126.

[125] See "DataFlow" on page 126.

[126] See "ActivitySetting" on page 121.

[127] See "ControlFlow" on page 126.

[128] See "DataFlow" on page 126.

[129] See "ActivityExtensionSetting" on page 122.

[130] See "Activity classification rules" on page 60.

[131] See "FDL ProgramActivity" on page 142.

Activity states

Description:

References to activity states "_Finished", "_ForceFinished", and "_Skipped" in a condition expression are not supported.

Explanation

Other than in WMQWF, WebSphere Process Server does not distinguish between states "finished" and "forcefinished". Hence, there is no need to refer to these states in an exit or transition condition, because the activity state will always be "finished" as soon as it can be referred to in a condition. If you want to refer to states "finished" or "skipped" of remote activities you must replace the corresponding conditions by equivalent Java expressions, because the generated XPath conditions do not support the required API calls.

Hint

But the migration of references to state "_Expired" in condition expressions is supported (see "Activity expiration" on page 75).

References to binary data in conditions

Description:

The FDL2BPEL Conversion Tool cannot migrate condition expressions that refer to binary data.

Hint

This limitation should be of minor importance, because binary data are intended for scanned or image data that would be rarely referred to in a condition expression.

Data flow

Prompting for data at process start

Description

The FDL2BPEL Conversion Tool ignores the process setting "Force prompt for data at process start".[132]

Explanation

A BPEL process does not offer an equivalent setting. The BPC explorer will always prompt you for process input data.

Initial process input data

Description

The setting of initial data values for the process input container cannot be migrated to BPEL.[132]

Explanation

The FDL2BPEL Conversion Tool maps the setting of initial container data values to a BPEL "Assign" activity that is executed right after the "Receive" activity. This "Assign" activity cannot set the initial process input data, because it would overwrite the process input data previously assigned by the "Receive" activity.[133]

Hint

You can write a Java snippet which sets initial data values for such data members that did not get a value while performing the "receive" activity that started the process.

[132] See "ProcessSetting" on page 114.

[133] See "Mapping FDL data flow to BPEL" on page 77.

Global container database settings

Description

The FDL2BPEL Conversion Tool does not migrate database settings for a global data container.[134] [135] Unsupported global container database settings comprise:

- no queries
- table name
- index settings

Explanation

Database settings for a WMQWF global data container have the following meaning.

No queries:

If selected, no queries are generated at runtime, although the global container is used for the process. This means that the global container data is available in the audit trail and you can also use the container API.

Table name:

You can define a name for the WMQWF database table. This may help you to reduce database resources by sharing the table among multiple process models.

Index settings:

To optimize performance of your WMQWF database, you can define index settings.

Note that WPS stores "query properties" in a single table for all process instances. You cannot customize this table with settings such as "table name" or "index settings" like in WMQWF.

Predefined data members

Description

BPEL does not support predefined data that is equivalent to the FDL predefined data members.

Explanation

BPEL has no equivalent concept of a "data container" that consists of user-defined data members and predefined data members. However, the FDL2BPEL Conversion Tool converts FDL data structures to corresponding message types that include the predefined data members like "_RC" and "_ACTIVITY_INFO.

Hint

You can run the FDL2BPEL Conversion Tool with option **-pi** that requests the initialization of the predefined data members _ACTIVITY and _PROCESS_MODEL. Option **-pn** disables the generation of predefined members if you do not need them.[136]

Subprocess invocation

Description

The input/output data for an FDL process activity and input/output data for any subprocesses must be identical before you can convert to BPEL.

For example, this is the situation if you have an FDL file with a process activity "My_Activity" that invokes a subprocess "My_Process" with a data structure that is inconsistent with this rule. After migrating the FDL, the WebSphere Integration Developer will display an error message like the following one:

[134] See "GlobalContainerSetting" on page 115.

[135] See "Queriable global data container" on page 55.

[136] This option is only available in the commandline mode.

"The messageType of the variable ‘My_Activity_IN' and the input element of operation ‘My_Process_OP' must be the same (activity 'My_Activity')."

Explanation

Unlike in FDL, the BPEL specification does not allow a "subset" or "superset" relationship between data passed from one process to another.

Hint

If you need such a combination you must either modify the FDL model before migration or insert s*nippets* with data mapping instructions to the BPEL model for after migration.

Data arrays

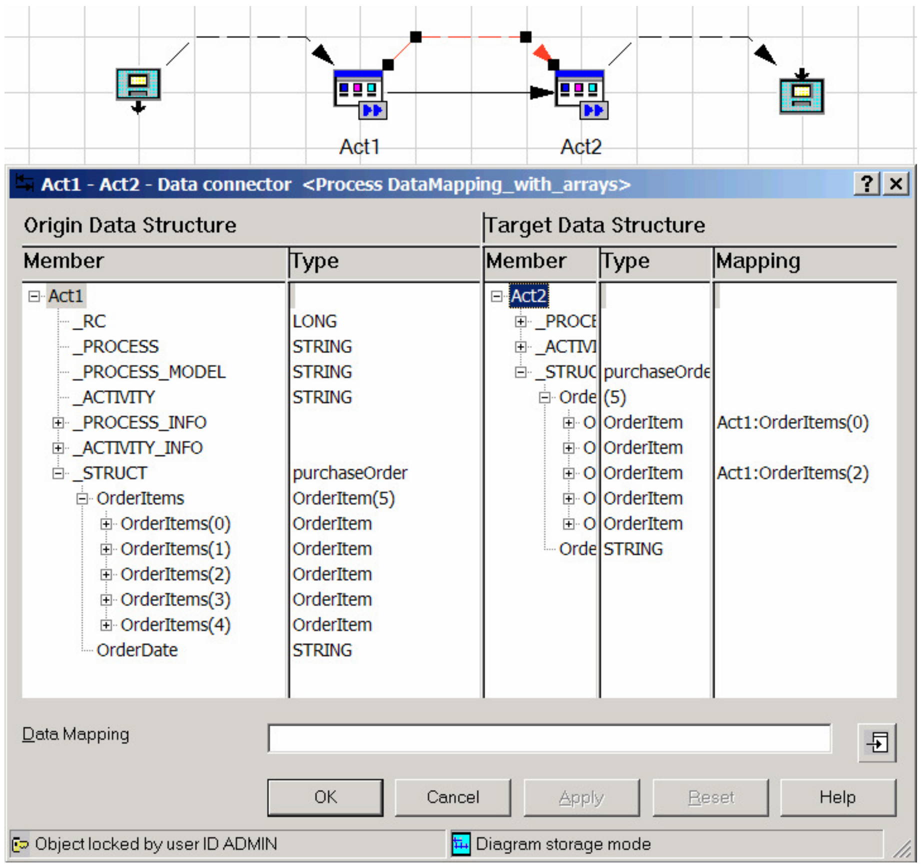
Description:

WPS cannot execute Java snippets that assign data values to array elements in non-ascending or interrupted index order.

Explanation:

The following example illustrates how the FDL2BPEL Conversion Tool translates an FDL data connector with data mappings of two array elements to a Java snippet:

WMQW Buildtime:


Act1
Act2
Act1 - Act2 - Data connector <Process DataMapping_with_arrays>
OK
Cancel
Apply
Reset
Help

Origin Data Structure

Member	Type
Act1	
_RC	LONG
_PROCESS	STRING
_PROCESS_MODEL	STRING
_ACTIVITY	STRING
_PROCESS_INFO	
_ACTIVITY_INFO	
_STRUCT	purchaseOrder
OrderItems	OrderItem(5)
OrderItems(0)	OrderItem
OrderItems(1)	OrderItem
OrderItems(2)	OrderItem
OrderItems(3)	OrderItem
OrderItems(4)	OrderItem
OrderDate	STRING

Target Data Structure

Member	Type	Mapping
Act2		
_PROCE		
_ACTIVI		
_STRUC	purchaseOrde	
Orde	(5)	
O	OrderItem	Act1:OrderItems(0)
O	OrderItem	
O	OrderItem	Act1:OrderItems(2)
O	OrderItem	
O	OrderItem	
Orde	STRING	

Data Mapping

Object locked by user ID ADMIN | Diagram storage mode

Figure 92: Mapping data array with interrupted index order

FDL source code:

```
DATA
  FROM 'Act1' TO 'Act2'
  MAP 'OrderItems(0)' TO 'OrderItems(0)'
  MAP 'OrderItems(2)' TO 'OrderItems(2)'
```

WID process editor:

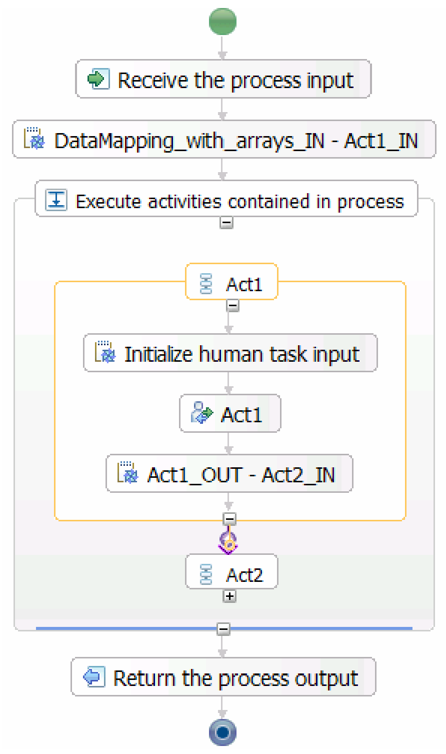

Receive the process input
DataMapping_with_arrays_IN - Act1_IN
Execute activities contained in process
Act1
Initialize human task input
Act1
Act1_OUT - Act2_IN
Act2
Return the process output

Figure 93: Snippet "Act1_OUT - Act2_IN" contains array mapping

Java code (in snippet "Act1_OUT – Act2_IN"):

```
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(
                                   Act1_OUT, "_STRUCT/OrderItems[1]",
                                   Act2_IN, "_STRUCT/OrderItems[1]",
                                   getVariableType("Act2_IN"));
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(
                                   Act1_OUT, "_STRUCT/OrderItems[3]",
                                   Act2_IN, "_STRUCT/OrderItems[3]",
                                   getVariableType("Act2_IN"));
…
```

At WPS runtime, this mapping will return an exception, because the mapping of the data array elements leaves element "_STRUCT/OrderItems[2]"[137] unset.

The reason is that the XPath expression *"_STRUCT/OrderItems[3]"* is equivalent to *"_STRUCT/OrderItems[position() = 3]"* and can only be interpreted as an array expression if all data elements with *position() < 3* exist.[138]

[137] Note that FDL expression "OrderItems(1)" corresponds to XPath expression "_STRUCT/OrderItems[2]".

Hint:

Check your WMQWF business model before migrating and ensure that all array mappings are in ascending and uninterrupted index order beginning with FDL index 1.

Staff assignment and notification

WMQWF staff repository

Description

The FDL2BPEL Conversion Tool ignores any staff definitions that may be contained in your FDL file (for example, "Person", "Role", "Organization", and "Level" definitions). [139]

Explanation

The target artifacts of the FDL2BPEL Conversion Tool cannot carry definitions that you might exploit for migrating your WMQWF staff repository.

Hint

If your target repository is Lightweight Directory Access Protocol (LDAP), you can use the WMQWF LDAP bridge for the migration of WMQWF staff repository.[140]

Notification

Description

Note that you cannot migrate the notification of an activity that is translated to an activity type other than "staff activity", according to the "activity classification rules (cp. section "Activity classification rules" on page 60).[141] [142]

Notification mode

Description

The FDL2BPEL Conversion Tool does not support the migration of notification mode settings.[142] [143] [144]

Explanation

The WMQWF setting of a notification mode affects the behavior of the expiration and notification timer if the process or an activity is in state "suspended":

- HOLD: The timers are stopped during the suspension time.
- RUN: The timers continue to run during the suspension time

Note that the FDL2BPEL Conversion maps WMQWF notification to an escalation for a human task in WebSphere Process Server.[145] There is no escalation mode available that is equivalent to "HOLD". If the task instance is suspended, the escalation timer keeps running.

[138] Note that **position()** returns values > 0, only.

[139] See "DeclarationAction" on page 104.

[140] See "MQ Workflow LDAP Bridge" on page 31 in IBM WebSphere MQ Workflow "Administration Guide", Version 3.6, SH12-6289

[141] See "ExplicitProcessStaffAssignmentSetting" on page 116.

[142] See "Notification" on page 124.

[143] See "DefaultProcessSetting" on page 107.

[144] See "ExplicitNotification" on page 125.

[145] See "Notification" on page 87.

Notification policy

Description

The FDL2BPEL Conversion Tool does not support the migration of notification policy settings "Assign substitute for notification if user is absent" and "Send second notification to same user".[146]

Explanation

There are no equivalent notification policies available in WebSphere Process Server.

Notification from container

Description

The FDL2BPEL Conversion Tool does not support the migration of dynamically assigned notification settings in the input container.[147]

Process notification

Description

WMQWF can be set up to notify a person if a specified maximum duration of a process instance is exceeded. The FDL2BPEL Conversion Tool does not support the migration of notification settings on the process level.[148]

Explanation

There is no equivalent notification function available in WPS.

Staff assignment by process category

Description

The FDL2BPEL Conversion Tool supports migrating the process category property. However, other than WMQWF, WPS does not support staff assignment by process category.[149] [150]

Hint

Possible migration solution:

1. Introduce a new role (group) that corresponds to the FDL process category.
2. Users previously authorized for the category must be reassigned to the new role (group).

Specify that "invocation task" or "administration task" that you define for the migrated process refers to the new "category" role (group).

Staff assignment from input container

Description

You cannot migrate staff assignments taken from the predefined data members in the input container.[151] [152]

[146] See "DefaultActivitySetting" on page 109.

[147] See "Notification" on page 87.

[148] See "ExplicitProcessStaffAssignmentSetting" on page 116.

[149] See "ProcessSetting" on page 114.

[150] See "Process category" on page 127.

[151] See "ProcessStaffAssignmentSetting" on page 115.

[152] See "ActivityStaffAssignmentSetting" on page 122.

Staff assignment criteria

Description

The FDL2BPEL Conversion Tool cannot map all staff assignment criteria that you specified for a WMQWF process or program activities that are contained in it.[153] [154] [155]

Explanation

WMQWF and the BPC Human Task Manager support different user registry systems and the staff query languages. Table 17 in section "Mapping WMQWF staff assignment criteria to TEL people assignment criteria", on page 85, lists the supported and unsupported mappings between WMQWF staff assignment criteria and TEL default people assignment criteria. There you find, for instance, that you cannot automatically migrate staff assignment criteria that implement the "Four eyes principle" ("exclude starter of activity").

Hint

For complex query logic, you might have to replace some of the generated people assignment criteria with user-defined people assignment criteria.

Staff assignment and notification criteria assigned to process activities

Description

The FDL2BPEL Conversion Tool ignores all staff assignment and notification criteria that you assigned to a process activity.[156]

Explanation

In WPS, you cannot specify either staff assignment or notification criteria for BPEL activities that invoke another process.

Program execution server

User-defined program execution server (UPES)

Description

The FDL2BPEL Conversion Tool does not migrate FDL server definitions, except for user-defined program execution server (UPES).[157] [158]

Explanation

The FDL2BPEL Conversion Tool does not migrate an FDL server definition, if the type is other than "USER_DEFINED_PROGRAM_EXECUTION_SERVER". For more details, see "WMQWF UPES migration details" on page 97.

Hints

If your WMQWF process model includes activities that communicate with a UPES, make sure that you select all referenced UPES definitions when you export the process model from WebSphere WMQWF Buildtime.

[153] See "DefaultActivitySetting" on page 109.

[154] See "ExplicitStaffAssignment" on page 123.

[155] See "OrgAssignment" on page 124.

[156] See "ProcessActivity" on page 119.

[157] See "Server" on page 109.

[158] See "UPESContext" on page 111.

During UPES migration you should check whether there are UPES applications that process messages which contain platform dependant "implementation data" (Note that WPS process will send "ActivityImplInvoke" messages without the element "ImplementationData").

XML message types supported by WPS UPES invocation

Description

The FDL2BPEL Conversion Tool migrates WMQWF program activities that invoke a UPES, provided that the UPES application does not depend on XML message types other than: [157]

• ActivityImplInvoke	Message received by a UPES
• ActivityImplInvokeResponse	Message returned by a UPES

Explanation

Note that other XML message types, such as "ActivityExpired" or "TerminateProgram", are not supported by the WMQWF data binding of WPS. This means that in the case of an activity expiration or the termination of a process instance a migrated UPES will not get notified via an XML message. For more details, see chapter "WMQWF UPES migration details" on page 97.

Hint

Ensure that the "ActivityImplInvokeResponse" messages returned by your UPES application always adhere to a correct syntax. Other than with WMQWF, you cannot expect that illegal messages are reported by means of "GeneralError" response from the WPS server.

Moreover, ensure that no UPES application depends on implementation or platform data elements of the XML messages, which are not supported.

Program execution unit taken from container

Description

The FDL2BPEL Conversion Tool does not migrate an FDL server definition that is dynamically referred to in the input container.

Model description and documentation fields

Data description and documentation

Description

The FDL2BPEL Conversion Tool does not migrate the data structure properties "description" and "documentation".[159]

Explanation

WPS has no "description" field that is equivalent to WMQWF.

Hint

Proposed solution: Concatenate WMQWF "description" and "documentation" (with inserted "titles" or "prefixes") and copy them to WPS "documentation".

Data member description and documentation

Description

The FDL2BPEL Conversion Tool does not migrate the data member properties "description" and "documentation".

[159] See "Data structure" on page 112.

Process category description and documentation

Description

The FDL2BPEL Conversion Tool does not migrate a description and documentation that you assigned to a process category.[160]

Process execution and monitoring

System network

Description

Migration is limited to the conversion of WMQWF business process to WPS business process. Network topology setting information (except for "UPES" definitions) is ignored.[161]

Explanation

MQ Workflow has a hierarchical structure to manage its networks. For example, the "domain" is the highest level in the hierarchy and can only contain one "system group". Each system group is made up of one or more "systems". Due to the different architecture, a migration of the WMQWF system network to WPS is not possible. Instead, set up WPS in a way that meets best your needs.

Properties inherited from domain or system

Description

You cannot migrate properties that your business process inherits from the domain or the system to which it belongs.[162]

Audit settings

Description

The migration of audit event definitions and settings of the WMQWF auditing system is not supported.[163] [164]

Hint

WebSphere Process Server offers a monitoring system. However, you must add the respective settings to your migrated business processes in a separate step.

Note that the FDL2BPEL Conversion Tool uses the FDL properties "AUDIT_FILTER_DB" and "AUDIT_FILTER_DB" as criteria to set the "business relevant" property in BPEL to "yes".[165] [166]

Process autonomy modes

Description

You cannot migrate process autonomy modes "full", "staff", "notification", and "administration".[167]

---

[160] See "Process category" on page 127.

[161] See "Topology" on page 105.

[162] See "TopologySetting" on page 106.

[163] See "DefaultProcessSetting" on page 107.

[164] See "DefaultActivitySetting" on page 109.

[165] See "Process property "business relevant"" on page 52.

[166] See "Activity property "business relevant"" on page 67.

[167] See "Autonomy" on page 108.

Explanation

WebSphere Process Server has no equivalent concept for the process autonomy modes "full", "staff", "notification", and "administration".

Hint

The FDL2BPEL Conversion Tool maps other autonomy settings, such that if the AUTONOMY definition in the FDL is "CONTROL", then the BPEL autonomy flag is set to "peer". In all other cases, it is set to "child".[168]

Program execution properties

Description

The FDL2BPEL Conversion Tool does not migrate the program execution properties of an FDL program definition.[169]

Explanation

The FDL2BPEL Conversion Tool uses FDL program definitions to generate corresponding interface definitions (WSDL "port types"[170]) for BPEL activities. The program execution properties have no equivalent usage, because WPS supports Service-Oriented Architecture, which is based on the concept of a "Web service" that serves as an abstraction from program execution details.

System properties assigned to process activities

Description

The FDL2BPEL Conversion Tool ignores FDL properties of a process activity that determine on which system a subprocess should run.[171]

Model accuracy

FDL ProgramActivity

Description

BPEL has no equivalent activity type with a similar semantic as the FDL "program activity" construct.

Explanation

BPEL does not permit the combination of the invocation of a program application with the assignment of a work item to a user. The FDL concept of a "program activity" means that the "program execution agent" (PEA) executes the application on the workstation of the user who claimed the respective workitem. With BPEL you cannot specify the location of the program execution as the location of the user.[172]

Hint

You can modify the generated BPEL files and insert additional "to-do tasks" or "invoke" activities according to your requirements. You can also insert additional synchronization points with user interaction. For example, inserting a "to-do task" that precedes an "invoke" activity so that the user can decide when the corresponding service is executed.

Example

[168] See "Process" on page 114.

[169] See "Program" on page 113.

[170] See "Mapping FDL PROGRAM to a port type" on page 47.

[171] See "ProcessActivitySetting" on page 119.

[172] See "ProgramActivity" on page 117.

Assume that the FDL2BPEL Conversion Tool performed the following translation of a UPES activity:

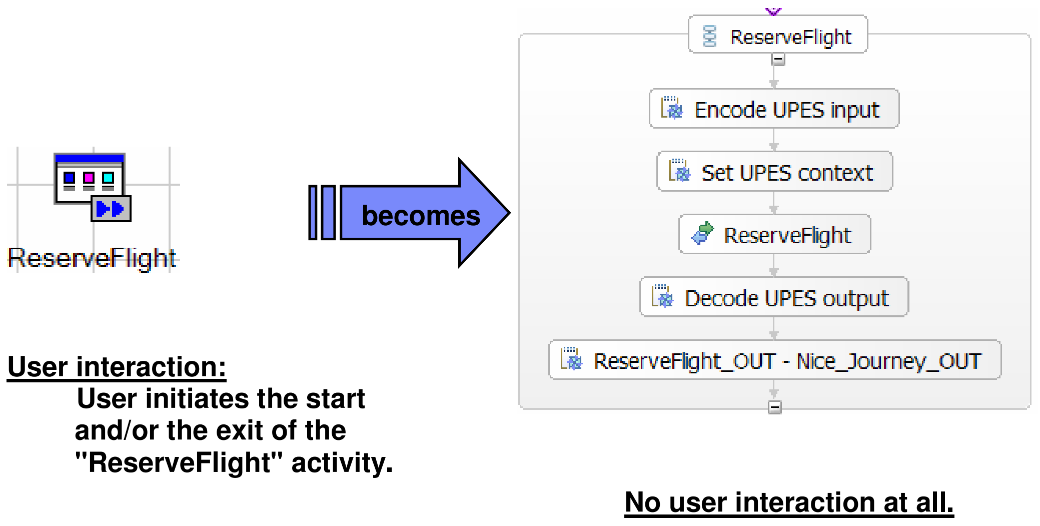

ReserveFlight
becomes
ReserveFlight
Encode UPES input
Set UPES context
ReserveFlight
Decode UPES output
ReserveFlight_OUT - Nice_Journey_OUT
User interaction:
User initiates the start and/or the exit of the "ReserveFlight" activity.
No user interaction at all.

Figure 94: User interaction is not translated to BPEL

You can add "Human Task" activities that let you control the UPES invocation using a manual start and exit:

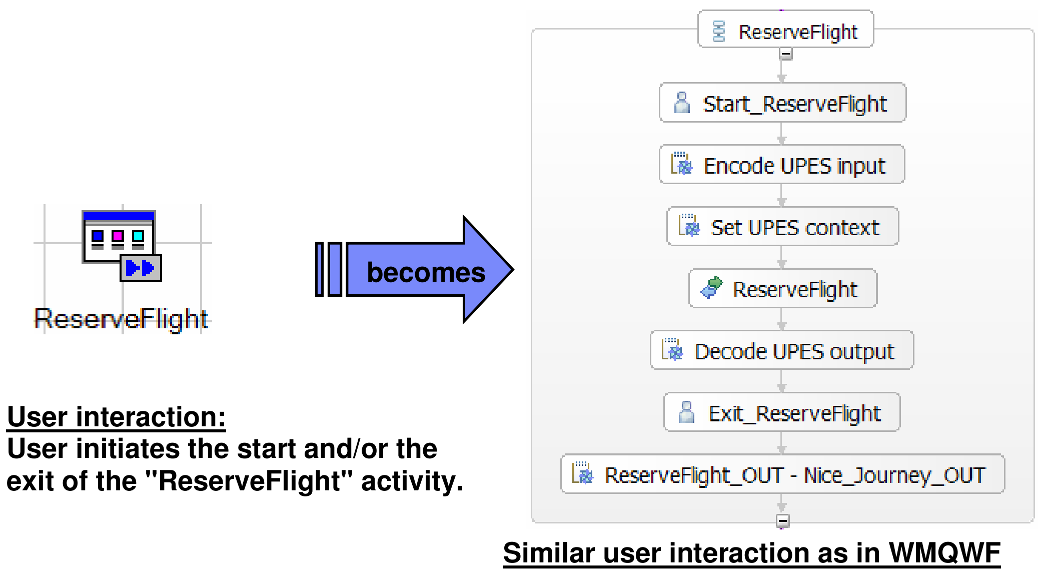


Figure 95: Inserting "Human Task" activities for manual activity start and exit

Workflow client

Support tools

Description

WMQWF permits the assignment of one or more additional programs (support tools) to program or process activities that can be started at runtime to help complete the current workitem. The FDL2BPEL Conversion Tool ignores the support tools specified in the FDL input file. [173] [174]

Explanation

WebSphere Process Server has no concept that is equivalent to a support tool.

Workitem refresh policy

Description

The FDL2BPEL Conversion Tool does not support the migration of a refresh policy. [175]

Explanation

The WMQWF refresh policy specifies how workitem lists are updated:

- PULL: In the client, a workitem list is only updated if the worklist is refreshed.
- PUSH: When used with the ActiveX client or Windows Runtime client, a worklist may be defined in "push" mode. In this case, workitem changes are sent to the client without a refresh initiated by the client. For all other clients, this option is not available.

Not that the refreshing behavior of the BPC explorer is equivalent to the "PULL"-policy.

Policy for how to deal with finished workitems

Description

The FDL2BPEL Conversion Tool does not support the WMQWF policy for how to deal with finished workitems. [175]

Policy for how to deal with finished process instances

Description

The FDL2BPEL Conversion Tool does not support the specification of a time period for the policy for how long finished process instances should be kept.[175] [176]

Hint

The FDL2BPEL Conversion Tool supports the settings "never" and "forever".

[173] See "DeclarationAction" on page 104.

[174] See "ActivityExtensionSetting" on page 122.

[175] See "DefaultProcessSetting" on page 107.

[176] See "TimePeriod" on page 129.