



WebSphere Process Server V6

FDL2BPEL Conversion

Version 6.2

Migration of WebSphere MQ Workflow source artifacts to WebSphere Process Server

Autor: Jens Grotrian (jgr@de.ibm.com)

Third Edition (December 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 Research & Development GmbH
Department 6111
Schoenaicher Strasse 220
D-71032 Boeblingen

Related Publications

IBM WebSphere Process Server

- WebSphere Integration Developer Version 6.2 Information Center at <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp>
- WebSphere Integration Developer Version 6.2 Information Center, Migrating from MQ Workflow at <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.wbit.620.help.migration.ui.doc/topics/tmqwf.html>
- 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>)

IBM Redbooks

- WebSphere MQ Workflow transition to WebSphere Process Server (available at <http://www.redbooks.ibm.com/redpieces/pdfs/sg247282.pdf>)

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

- IBM
- WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

CONTENTS

Figures	8
Tables	10
Abbreviations	11
What is new with Version 6.2?	12
Optimization options	12
Administration task for activity repair scenarios	12
More natural reference to BPEL variables in XPath expressions	12
Improved mapping accuracy	12
CHAPTER 1: INTRODUCTION.....	13
Subject of this document.....	13
Who are the readers that will most benefit from this document?	13
Migrating with or without a tool?	13
Are the migrated models too complex?	13
FDL2BPEL usage modes	15
What FDL2BPEL Conversion cannot do for you.....	15
Some best practice hints	15
Mapping rules (overview)	15
CHAPTER 2: USING FDL2BPEL CONVERSION	17
Exporting a WMQWF model from WMQWF Buildtime	17
Using as WID Migration Wizard.....	18
Using as command line tool	24
Prerequisites	24
How to translate WMQWF models into BPC models.....	24
Running FDL2BPEL Conversion	24
Using optimization options	26
Importing the artifacts into WID.....	27
CHAPTER 3: UNDERSTANDING THE GENERATED FILES	30
Introduction	30
The XSD file	30
The WSDL file	31
The BPEL files	31
The BPELEX files	31
The TEL files	31
The COMPONENT files	31
The IMPORT files	31
The JAVA files	32
The MON file	32

CHAPTER 4: MIGRATION MAPPING RULES	33
Mapping of FDL names to BPEL.....	33
Introduction	33
Name mapping rules in detail.....	33
FDL to XSD mapping rules	34
Mapping FDL "STRUCTURE" to a "Business Object"	34
Mapping an FDL data container to an XML schema definition	37
Mapping the data exchanged with a UPES	40
FDL to WSDL mapping rules.....	42
Generation of partner link type definitions	42
Mapping FDL PROCESS to a partner link type	42
Mapping FDL SERVER to a BPEL partner link type	42
Mapping FDL container data to WSDL message definitions	43
Mapping FDL interaction interfaces to WSDL port types.....	43
Mapping FDL PROCESS to a port type.....	43
Mapping FDL PROGRAM to a port type.....	44
Mapping FDL SERVER to a port type for a UPES	47
FDL to BPEL mapping rules.....	48
Mapping FDL PROCESS to BPEL	48
Process property "business relevant"	49
Process autonomy	50
Process category	50
Partner links	51
Mapping FDL data container to BPEL.....	51
<i>Queryable global data container</i>	<i>52</i>
Mapping the FDL workflow to BPEL.....	54
Mapping WMQWF activities to BPEL.....	56
Mapping FDL PROGRAM_ACTIVITY to BPEL	56
<i>Activity classification rules.....</i>	<i>57</i>
<i>Empty activity.....</i>	<i>58</i>
<i>Service invocation activity (UPES activity).....</i>	<i>58</i>
<i>Staff activity</i>	<i>62</i>
<i>Activity property "business relevant".....</i>	<i>63</i>
Mapping FDL BLOCK to BPEL.....	63
Mapping FDL PROCESS_ACTIVITY to BPEL.....	64
<i>Static subprocess invocation.....</i>	<i>64</i>
<i>Late binding of subprocess names</i>	<i>65</i>
Mapping WMQWF control flow to BPEL.....	67
Transition conditions	67
Start conditions.....	68
Exit conditions	69
Activity expiration	71
Mapping FDL data flow to BPEL	73
Mapping WMQWF staff assignment criteria to TEL people assignment criteria.....	80
Introduction	80
Mapping limitations	81
Staff repository	81
Using default people assignment criteria.....	81
Distinction between "UPES activities" and "staff activities"	82
Migration of staff assignment definitions to be inherited	82
Mapping of staff assignment criteria.....	82
Notification	83
Substitution	86
Mapping FDL to Service Component Architecture	87
SCA Components	88

SCA Imports	89
Data binding type	89
Endpoint configuration	91
Preferred interaction style	94
CHAPTER 5: WMQWF UPES MIGRATION DETAILS	95
CHAPTER 6: OPTIMIZING THE MIGRATED BUSINESS MODEL	96
Optimizing BPEL process models	96
Merging Java snippets	96
Merging parallel Java snippets	96
Merging sequential Java snippets	97
Removing unnecessary structural elements	98
Sharing variables	100
Overview	100
Example	101
Using additional optimization settings (for experts only)	103
APPENDIX 1: MIGRATION COVERAGE LISTED BY FDL SYNTAX EXPRESSIONS	105
Introduction	105
FDL source file	105
DeclarationAction	105
Topology	106
System	106
TopologySetting	107
DefaultProcessSetting	108
Autonomy	109
DefaultActivitySetting	110
Server	110
ServerSetting	111
UPESContext	112
ProcessModeling	112
Data structure	113
Program	114
Process	115
ProcessSetting	115
GlobalContainerSetting	116
ProcessStaffAssignmentSetting	116
ExplicitProcessStaffAssignmentSetting	117
Construct	117
Activity	118
ProgramActivity	118
ProgramActivitySetting	119
ProcessActivity	120
ProcessActivitySetting	120
Block	121
BlockSetting	121
ActivitySetting	122
ActivityExtensionSetting	123

ActivityStaffAssignmentSetting	123
ExplicitStaffAssignment	124
OrgAssignment	125
Notification	125
ExplicitNotification	126
Expiration	126
<i>ControlFlow</i>	127
<i>DataFlow</i>	127
Process category	128
Conditions	129
<i>Condition</i>	129
Common variables	130
TimePeriod	130
APPENDIX 2: MIGRATION HINTS	131
FDL source file	131
FDL with codepage different from system codepage	131
Migration of early FDL versions	131
FDL processing actions	131
Problem detection	132
Graphical business process editor	132
Migration of WMQWF Buildtime tool set definitions not supported	132
Graphical layout information	132
Activity icons	133
Control flow	133
Activity start and exit mode	133
Activity states	133
References to binary data in conditions	134
Data flow	134
Prompting for data at process start	134
Global container database settings	134
Predefined data members	135
Subprocess invocation	135
Data arrays	135
Staff assignment and notification	138
WMQWF staff repository	138
Notification	138
Notification mode	138
Notification policy	138
Notification from container	139
Process notification	139
Staff assignment by process category	139
Dynamic staff assignment from input container	141
Staff assignment criteria	141
Staff assignment and notification criteria assigned to process and UPES activities	142
Migration of "Exclude starter of activity"	142
Program execution server	143
User-defined program execution server (UPES)	143
XML message types supported by WPS UPES invocation	143
Program execution unit taken from container	143
Model description and documentation fields	144
Data description and documentation	144
Data member description and documentation	144
Process category description and documentation	144

Process execution and monitoring	144
System network	144
Properties inherited from domain or system	144
Audit settings	145
Process autonomy modes	145
Program execution properties	145
System properties assigned to process activities	145
Model accuracy	146
FDL program activity	146
Workflow client	147
Support tools	147
Workitem refresh policy	147
Policy for how to deal with finished workitems	147
Policy for how to deal with finished process instances	148

Figures

Figure 1: Topology preservation policy (1)	14
Figure 2: Topology preservation policy (2)	14
Figure 3: Exporting a WMQWF model from WMQWF Buildtime	18
Figure 4: Running FDL2BPEL as WID Migration Wizard (1)	19
Figure 5: Running FDL2BPEL as WID Migration Wizard (2)	20
Figure 6: Running FDL2BPEL as WID Migration Wizard (3)	21
Figure 7: Running FDL2BPEL as WID Migration Wizard (4)	22
Figure 8: Running FDL2BPEL as WID Migration Wizard (5)	23
Figure 9: Running FDL2BPEL as WID Migration Wizard (6)	24
Figure 10: Running FDL2BPEL from the command line	26
Figure 11: Creating a new business integration module	28
Figure 12: Importing FDL2BPEL generated files (1)	28
Figure 13: Importing FDL2BPEL generated files (2)	29
Figure 14: Sample FDL data structure	34
Figure 15: WID Business Object Editor	36
Figure 16: Hierarchical FDL data structure	36
Figure 17: Hierarchically structured Business Object	37
Figure 18: FDL data container	38
Figure 19: Mapping FDL data container to Business Object.....	39
Figure 20: UPES input message as Business Object	41
Figure 21: UPES output message as Business Object	41
Figure 22: Process interface.....	44
Figure 23: Program requires these data structures	44
Figure 24: Interface with single operation	45
Figure 25: Program can handle any data structures	45
Figure 26: Interface with multiple operations.....	46
Figure 27: Asynchronous execution of UPES	47
Figure 28: UPES interface.....	47
Figure 29: Opening the business process editor	48
Figure 30: Mapping process properties (1)	48
Figure 31: Mapping process properties (2)	49
Figure 32: Mapping process properties (3)	49
Figure 33: Mapping process properties (4)	50
Figure 34: Process Category	50
Figure 35: Partner links	51
Figure 36: Mapping FDL data containers to BPEL variables	52
Figure 37: Mapping Global Container to business object	53
Figure 38: Mapping FDL data structure to BPEL query properties	54
Figure 39: Mapping the workflow structure (1)	54
Figure 40: Mapping the workflow structure (2)	55
Figure 41: Mapping the workflow structure (3)	56
Figure 42: Assumed "staff activity"	57

Figure 43: Activity classification rule: "empty activity"	58
Figure 44: Activity classification rule: "service invocation activity"	59
Figure 45: Encoding the UPES input	60
Figure 46: Decoding the UPES output	60
Figure 47: Setting the UPES context	61
Figure 48: Classification rule "staff activity"	62
Figure 49: WID human task editor	63
Figure 50: Mapping activity property "business relevant"	63
Figure 51: Mapping an FDL block to BPEL	64
Figure 52: Mapping an FDL subprocess call to BPEL	64
Figure 53: Subprocess invoke properties	65
Figure 54: Process from container	65
Figure 55: Late-bound subprocess invocation	66
Figure 56: Assigning the subprocess name	67
Figure 57: Mapping a transition condition expression to BPEL	68
Figure 58: Mapping a start condition expression to BPEL	69
Figure 59: Mapping an exit condition expression to BPEL	71
Figure 60: Activity expiration	72
Figure 61: BPEL representation of FDL activity "Get customer name"	73
Figure 62: Translated exit condition of FDL activity "Get customer name"	73
Figure 63: BPEL representation of FDL activity "Check customer data"	73
Figure 64: Translated transition condition referring to activity "Check customer data"	73
Figure 65: FDL data flow	74
Figure 66: BPEL representation of FDL data flow (1)	74
Figure 67: BPEL representation of FDL data flow (2)	75
Figure 68: Mapping FDL data mapping to a Java snippet	76
Figure 69: First and second notification of an FDL activity	84
Figure 70: Mapping first notification to an escalation step of a human task	85
Figure 71: Mapping second notification to an escalation step of a human task	86
Figure 72: FDL process "Nice Journey"	87
Figure 73: FDL subprocess "Book Car"	88
Figure 74: WID assembly diagram view	88
Figure 75: A deliberately omitted wire in the assembly diagram	89
Figure 76: How BPC communicates with a UPES	90
Figure 77: MQ JMS binding of an SCA import	91
Figure 78: Java class implementing the data binding	91
Figure 79: Queue manager in WMQWF Buildtime	92
Figure 80: Queue manager in the WID assembly editor (properties view)	92
Figure 81: Request queue in the WID assembly editor (properties view)	93
Figure 82: Response queue in the WID assembly editor (properties view)	93
Figure 83: Preferred interaction style of UPES interface	94
Figure 84: Distributing the activity output data with a merged Java snippet	97
Figure 85: Merging sequentially arranged Java snippets	98
Figure 86: Simply linked activity chain (WMQWF Buildtime)	98
Figure 87: Simply linked activity chain (WID process editor)	99
Figure 88: BPEL process after using option "Remove unnecessary structural elements"	100
Figure 89: Sharing of BPEL variables	100
Figure 90: WMQWF process with two simple activity chains	102
Figure 91: Migrated activity chains without optimization	102
Figure 92: Migrated activity chains with shared BPEL variables	103
Figure 93: Mapping data array with interrupted index order	136
Figure 94: Snippet "Act1_OUT - Act2_IN" contains array mapping	137
Figure 95: Creating a human task for the potential starters of the migrated process	140
Figure 96: Specifying the potential starters as group members	141
Figure 97: User interaction is not translated to BPEL	146
Figure 98: Inserting "Human Task" activities for manual activity start and exit	147

Tables

Table 1: Mapping rules (overview)	16
Table 2: General FDL2BPEL command line options	25
Table 3: FDL2BPEL optimization options.....	27
Table 4: Additional FDL2BPEL optimization options.....	27
Table 5: Files generated by FDL2BPEL.....	30
Table 6: Mapping basic FDL data types.....	35
Table 7: Activity classification rules.....	57
Table 8: UPES context data	61
Table 9: Mapping an exit condition expression to BPEL.....	70
Table 10: Semantic rules of "merge" operation.....	78
Table 11: Example 1 ("Merge" operation with two or more inbound data connectors).....	78
Table 12: Example 2 ("Merge" operation with inbound data connector leading to an input container with default values).....	78
Table 13: Example 3 ("Merge" operation with inbound data connector and data loop connector)	79
Table 14: Example 4 (Combining examples 1 – 3)	79
Table 15: Example 5 (UPES or PEA activity ("staff activity") with default output data and/or default data connector).....	79
Table 16: Example 6 (Process activity with default output data and/or default data connector)	79
Table 17: Example 7 (Block activity with default output data and/or default data connector)..	80
Table 18: Comparing staff/people assignment criteria (WMQWF vs. TEL)	81
Table 19: Mapping WMQWF staff assignment criteria to TEL default people assignment criteria.....	83
Table 20: Substitution mapping rules	86
Table 21: Queue naming conventions.....	94
Table 22: UPES execution mode	94

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 6.2?

Optimization options

Former versions of FDL2BPEL Conversion translated FDL to BPEL business models strictly adhering to a **topology preservation policy**. This strategy ensures that the migrated BPEL business model has a structure that is similar to the originating FDL business model. Similar model structures make it less difficult to become familiar with the new appearance of the BPEL model and to validate its correctness. A disadvantage of preserving the topology is that the migrated BPEL model possibly contains dispensable activity nodes and variables. FDL2BPEL Conversion V6.2 supports new optimization options that let you reduce the number of activity nodes and variables thereby improving the runtime performance of the migrated model.¹

Administration task for activity repair scenarios

An administration task is created to enable dedicated staff via work items to repair "invoke" activities (e.g. forceRetry(), forceComplete()). In case that in the MQWF model "all people" is specified for the activity repair (that means no DONE_BY statement in the related FDL part) then an administration task is generated for the process administrator only.

More natural reference to BPEL variables in XPath expressions

FDL2BPEL Conversion translates exit conditions of FDL activities and transition conditions of FDL control connectors to XPath expressions of corresponding conditions in the BPEL model. Translating a reference to a container data member required to use the **getVariableData** XPath 1.0 function to access the corresponding data in a BPEL variable. Mapping BPEL variables directly to XPath 1.0 variables allows to get rid of the **getVariableData** function and to provide a more readable expression.

Improved mapping accuracy

Compared with former versions FDL2BPEL Conversion exhibits an improved coverage of mapping FDL to BPEL models. Here are some examples of resolved limitations:

- Initial process input data (The setting of initial data values for the process input container)
- Staff assignment from input container (Staff assignment criteria taken from predefined data member **Organization**)²
- Data and control connector names
- Staff assignment criteria (Option "Exclude starter of activity", which is also known as the "4-eyes principle").³

¹ See chapter "Optimizing the migrated business model" on page 96.

² See migration hint "Dynamic staff assignment from input container" on page 141.

³ See migration hint "Migration of "Exclude starter of activity"" on page 142.

Chapter 1: Introduction

Subject of this document

This document describes how to use FDL2BPEL Conversion Version 6.2. 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 6.2 (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 FDL2BPEL Conversion 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 FDL2BPEL Conversion 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 FDL2BPEL Conversion 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. FDL2BPEL Conversion translates FDL data connectors into "Snippet" activities.⁴
- BPEL does not know the concept of an activity data input/output container with default value settings. FDL2BPEL Conversion translates each data input/output container

⁴ See "Mapping FDL data flow to BPEL" on page 73.

into corresponding BPEL variables and the setting of default values into an "Assign" activity with name "Set the data default values".⁵

- 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.^{6 7}

In conclusion, your impression of increased complexity seems to be confirmed. However, the mapping rules of FDL2BPEL Conversion 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:

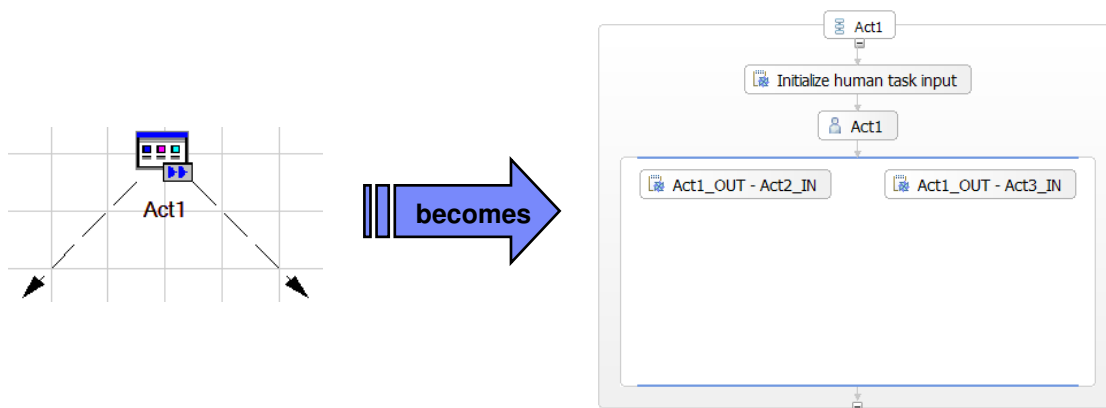



Figure 1: Topology preservation policy (1)

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 . Now you should see that the migrated business process contains the same number and sequence of activities:

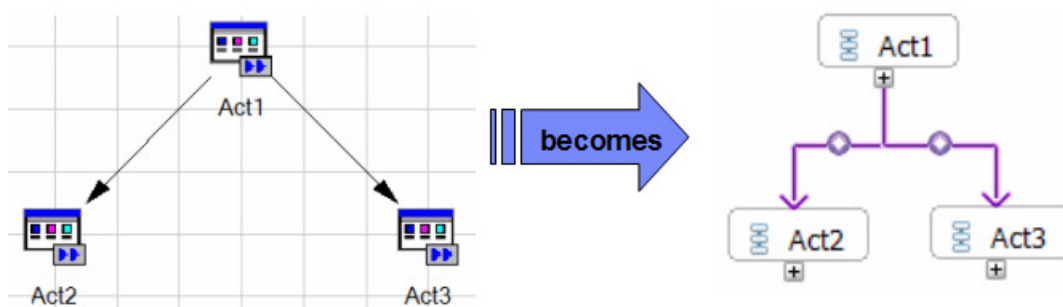


Figure 2: Topology preservation policy (2)

⁵ See "Mapping FDL data container to BPEL" on page 51.

⁶ See "Mapping the FDL workflow to BPEL" on page 54.

⁷ See "Exit conditions" on page 69.

FDL2BPEL usage modes

FDL2BPEL Conversion Version 6.2 is available in two modes:

1. WID Migration Wizard (see page 18)
2. Command line tool (see page 24)

Both modes of FDL2BPEL Conversion 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 FDL2BPEL Conversion cannot do for you

FDL2BPEL Conversion does not cover the migration of the following:

- WMQWF runtime instances
- WMQWF program applications that are invoked by a WMQWF *Program 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 migration hint "Migration of early FDL versions" on page 131).

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".⁸
- 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. FDL2BPEL Conversion automatically converts problematic FDL names to valid BPEL names.

FDL2BPEL Conversion 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:

⁸ See "Mapping WMQWF staff assignment criteria to TEL people assignment criteria" on page 80.

Table 1: Mapping rules (overview)

	WMQWF Process Model Construct	BPEL with Extensions Construct⁹
Structure	Process	<i>Process with execution mode: longRunning</i> (BPEL extension) <i>Partner links</i> for inbound and outbound interfaces of process
	Program activity	<i>Invoke</i> activity
	Process activity	<i>Invoke</i> activity
	Empty activity	<i>Empty</i> activity
	Block	<i>Parallel activities (also called flow activity)</i>
Control Flow	Start condition of activity	<i>Join condition</i> of activity
	Exit condition of activity	<i>While</i> activity (enclosing the actual activity)
	Control connector	<i>Link, source</i> element of activity, <i>target</i> element of activity
	Transition condition (of control connector)	<i>Transition condition</i> (assigned to <i>source</i> element)
	Staff assignment of activity	<i>Human task</i> activity (BPEL extension using assigned TEL task definition)
	Notification	<i>Escalation</i> (BPEL extension using assigned TEL task definition)
	Expiration for activity	<i>Expiration</i> for <i>invoke</i> activity (BPEL extension)
Data Flow	Source and sink	<i>Variables</i> for process input and process output <i>Receive</i> activity and <i>reply</i> activity
	Input / output container of activity	<i>Variables</i> for specification of input/output of <i>invoke</i> activity
	Data connector	<i>Snippet</i> (activity with inline Java code that copies data from a source BPEL variable to a target BPEL variable)
	Default data settings	<i>Assign</i> activity (assigns default data as literal values to a BPEL variable)
	Global data container	<i>Variable</i>

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. FDL2BPEL Conversion 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.

⁹ 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 FDL2BPEL Conversion

As already mentioned, you can migrate WMQWF business process models either by means of the WID Migration Wizard or by starting FDL2BPEL Conversion from a command line window. The first section of this chapter describes how to prepare an FDL file that you need for the migration task. The next subchapters explain the details of the two available usage modes.

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.^{10 11} Also note that FDL2BPEL Conversion expects a syntactically correct and complete FDL input file.¹² If the tool encounters severe errors, it will report them without starting the migration operation.

¹⁰ FDL files exported from WBI Modeler should be imported into WMQWF Buildtime before.

¹¹ See "FDL processing actions" on page 131.

¹² See "Problem detection" on page 132.

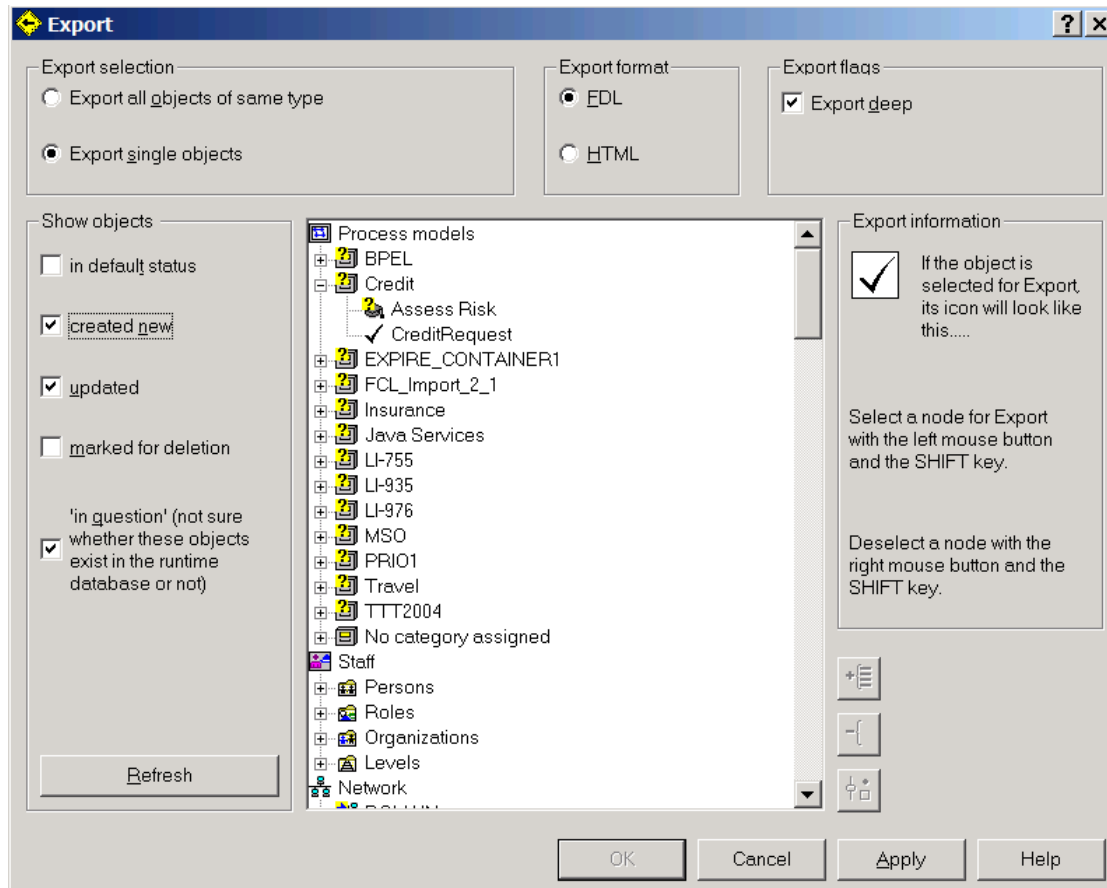


Figure 3: Exporting a WMQWF model from WMQWF Buildtime

Using as WID Migration Wizard

By importing a WMQWF business process model exported from WMQWF Buildtime¹³ 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 30). 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 17).
2. Start the WID.
3. In WID, invoke the wizard by selecting **File > Import... > Business Integration > WebSphere MQ Workflow FDL File**

¹³ See "Exporting a WMQWF model from WMQWF Buildtime" on page 17.

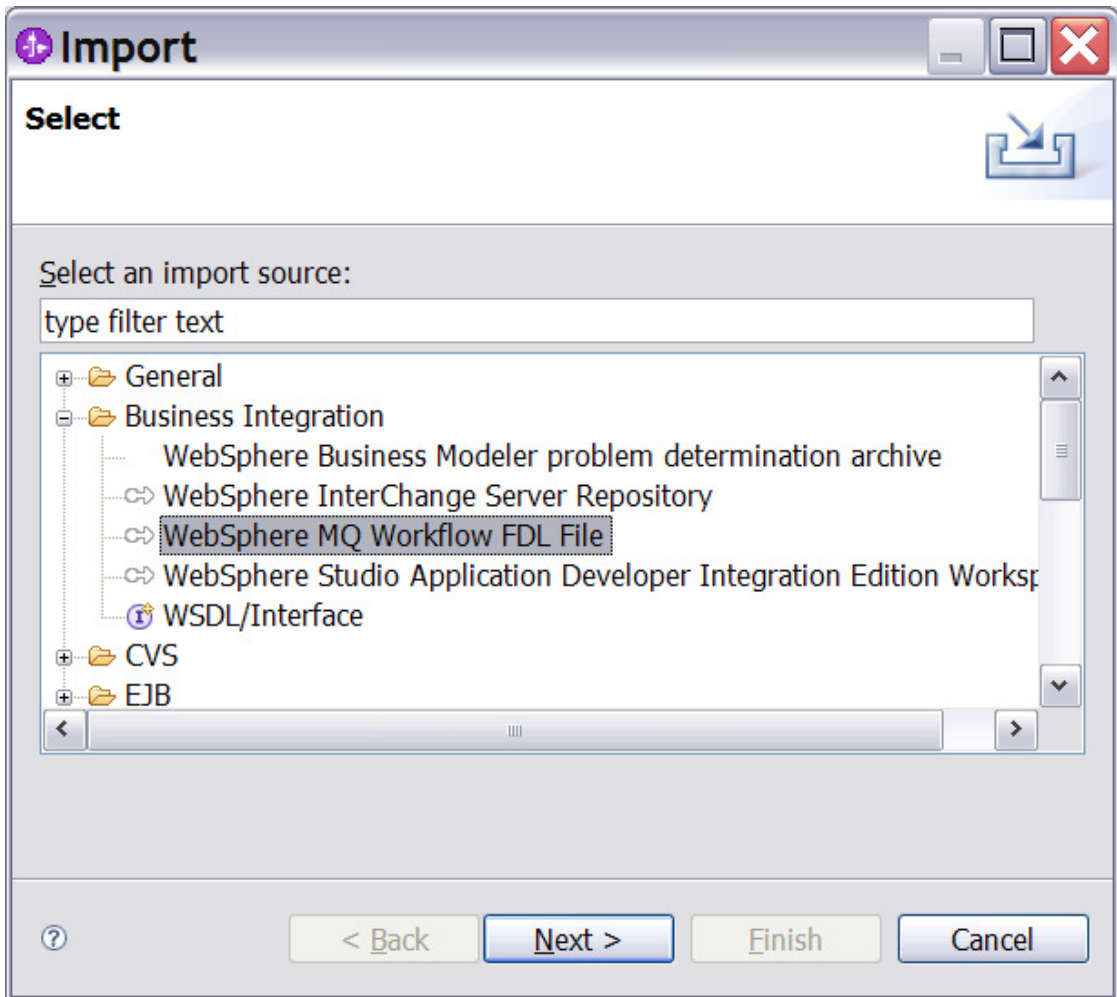


Figure 4: Running FDL2BPEL as WID Migration Wizard (1)

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.¹⁴ Enter the module name in the Module name field, and then click **Next >**.

¹⁴ This step corresponds to entering the **-i** argument if you start FDL2BPEL Conversion from the command line.

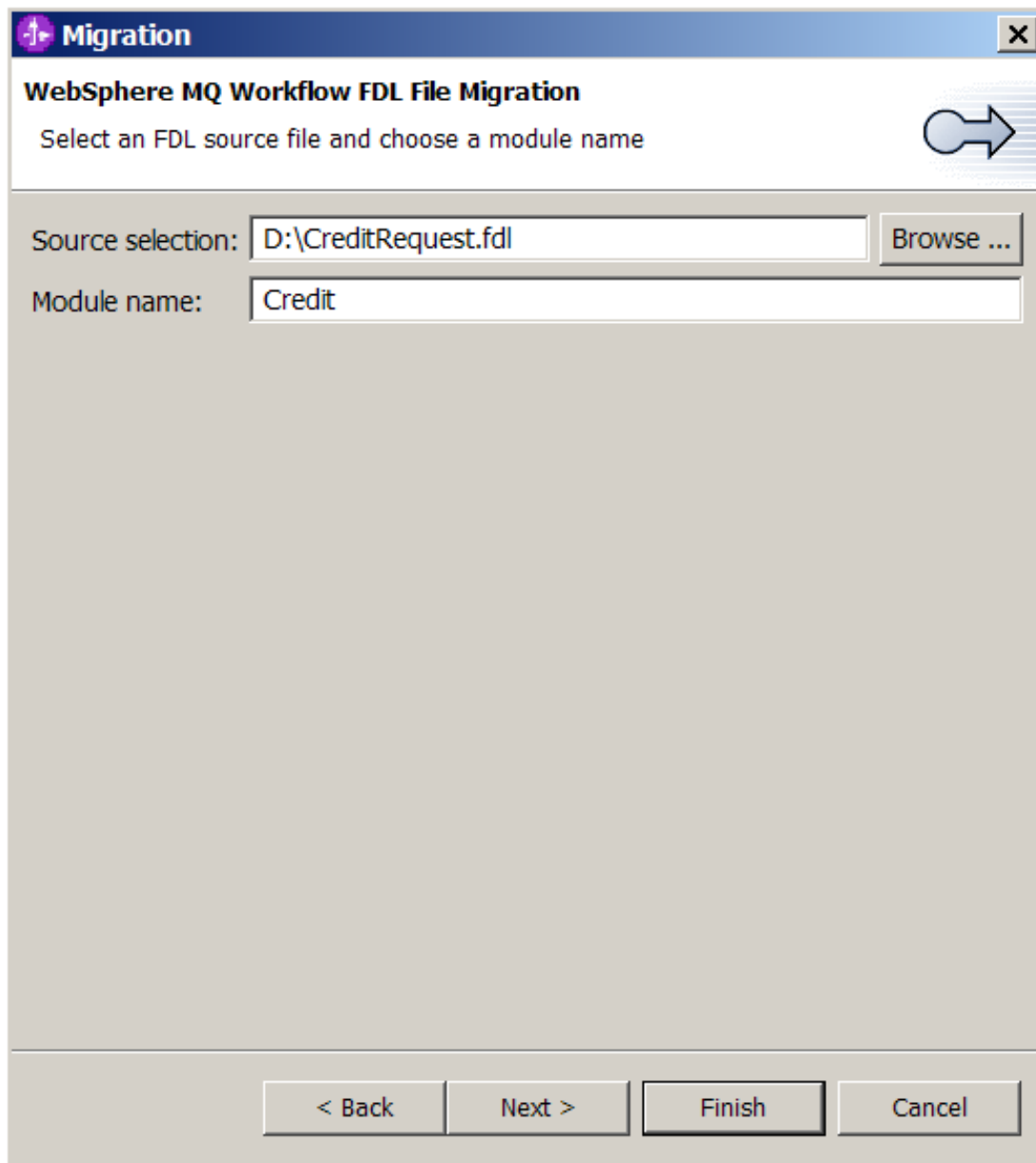


Figure 5: Running FDL2BPEL as WID Migration Wizard (2)

5. The "Migration Options" page for the "Artifact creation settings" 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.¹⁵ Otherwise, FDL2BPEL will solve any name conflicts automatically. If you have no need for using predefined data members, deselect option "Create predefined data members".¹⁶ The "Initialize predefined data members" check box adds extra nodes to the process to initialize predefined data members¹⁷. Moreover, you can overwrite the target namespaces for XML Schema¹⁸, WSDL¹⁹, and BPEL²⁰ definitions. Click **Next**.

¹⁵ This check box corresponds to the **-fc** option if you start FDL2BPEL Conversion from the command line.

¹⁶ This check box corresponds to the **-pn** option if you start FDL2BPEL Conversion from the command line.

¹⁷ This check box corresponds to the **-pi** option if you start FDL2BPEL Conversion from the command line.

¹⁸ This entry field corresponds to the **-tx** argument if you start FDL2BPEL Conversion from the command line.

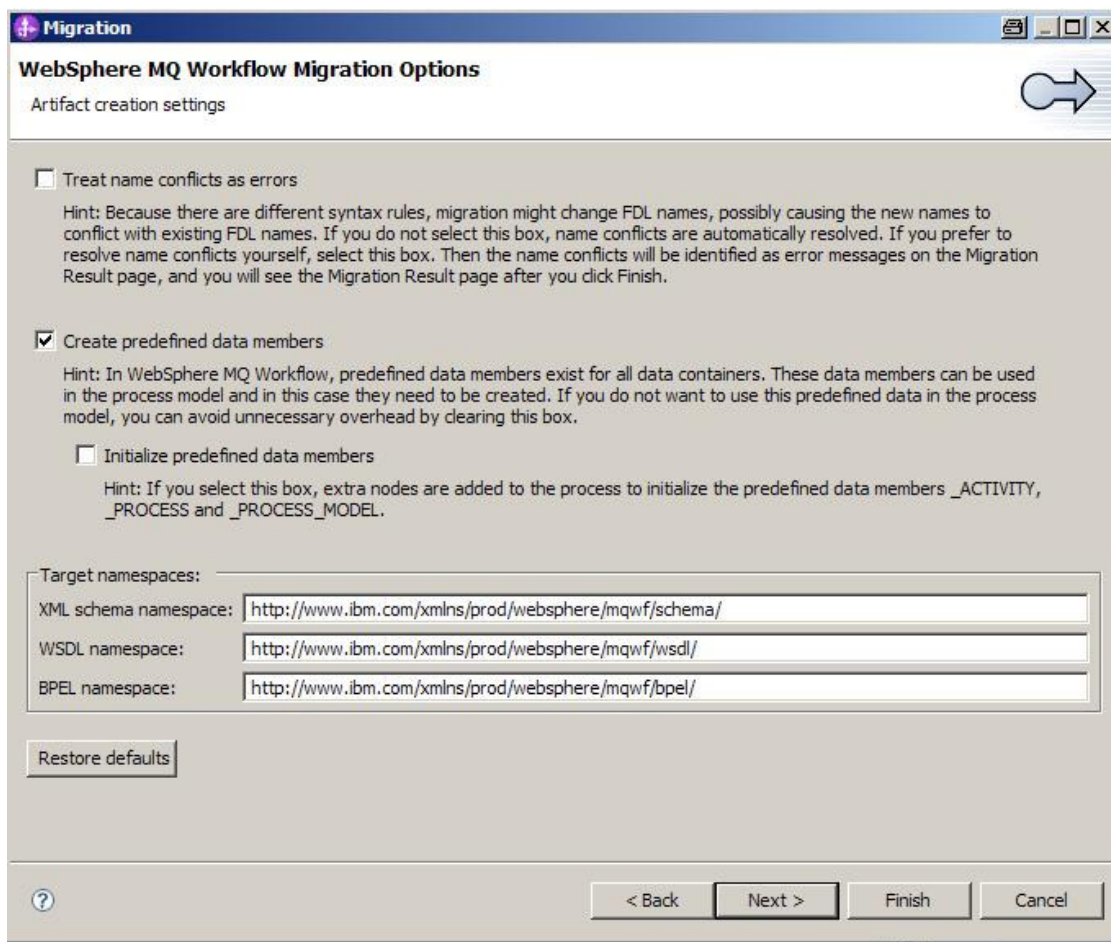


Figure 6: Running FDL2BPEL as WID Migration Wizard (3)

6. The "Migration Options" page for the optimization of the migrated business model opens. Select the appropriate options to merge Java snippets or to remove unnecessary structural elements or to allow sharing variables. Click **Finish**.

¹⁹ This entry field corresponds to the **-tw** argument if you start FDL2BPEL Conversion from the command line.

²⁰ This entry field corresponds to the **-tb** argument if you start FDL2BPEL Conversion from the command line.

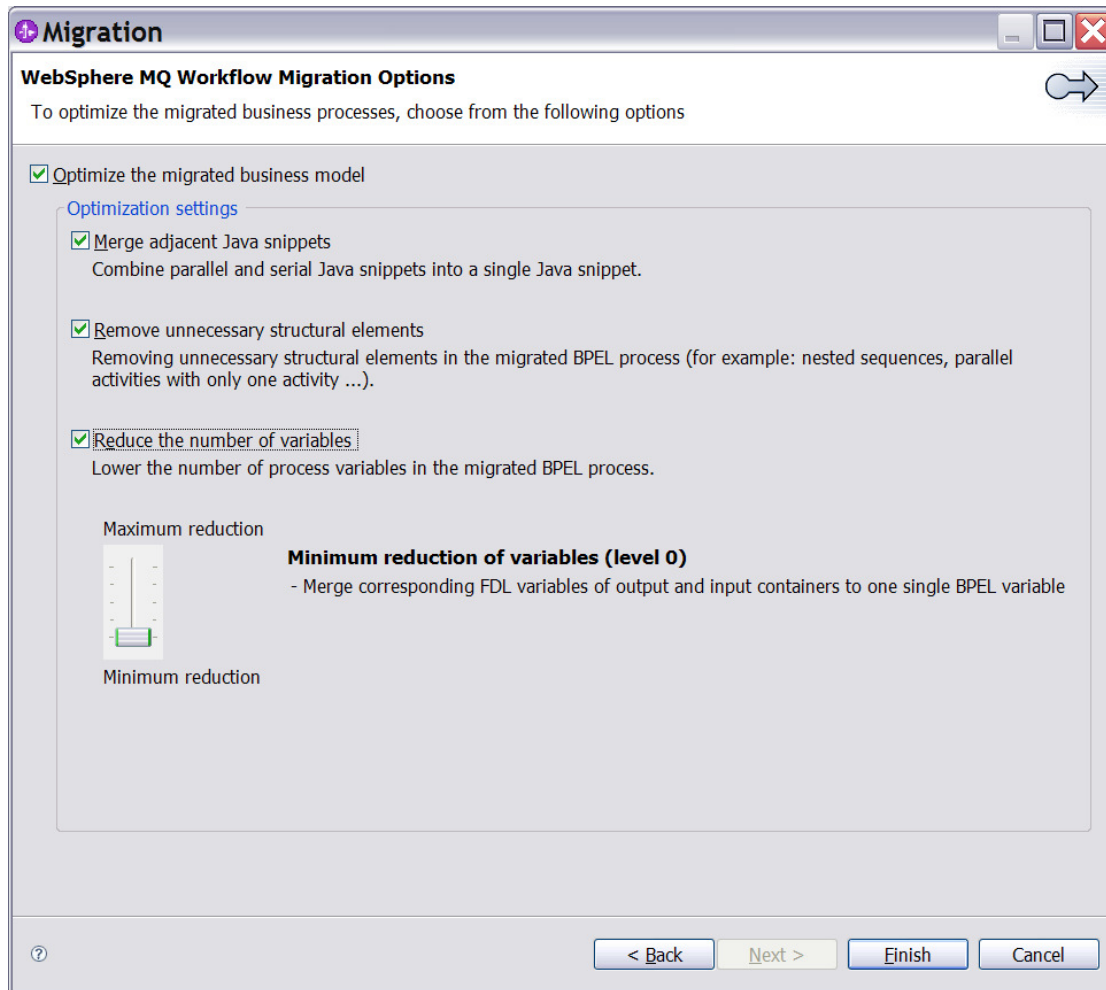


Figure 7: Running FDL2BPEL as WID Migration Wizard (4)

7. 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:

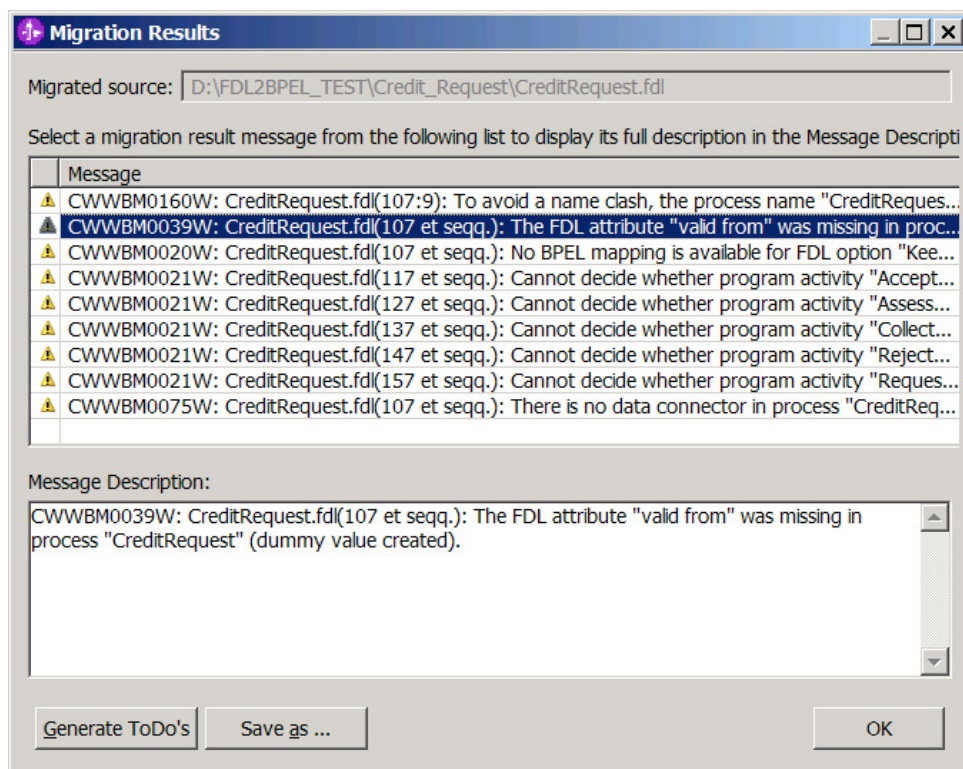


Figure 8: Running FDL2BPEL as WID Migration Wizard (5)

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 its description in the lower "Message Description" window.²¹ 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**.

8. 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 window²² you will find a new module in the Business Integration tree view with the name that you entered in step 4:

²¹ For the complete information about that message consult the WPS information center at <http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp> where you can find a reference chapter about all messages.

²² Warnings are allowed.

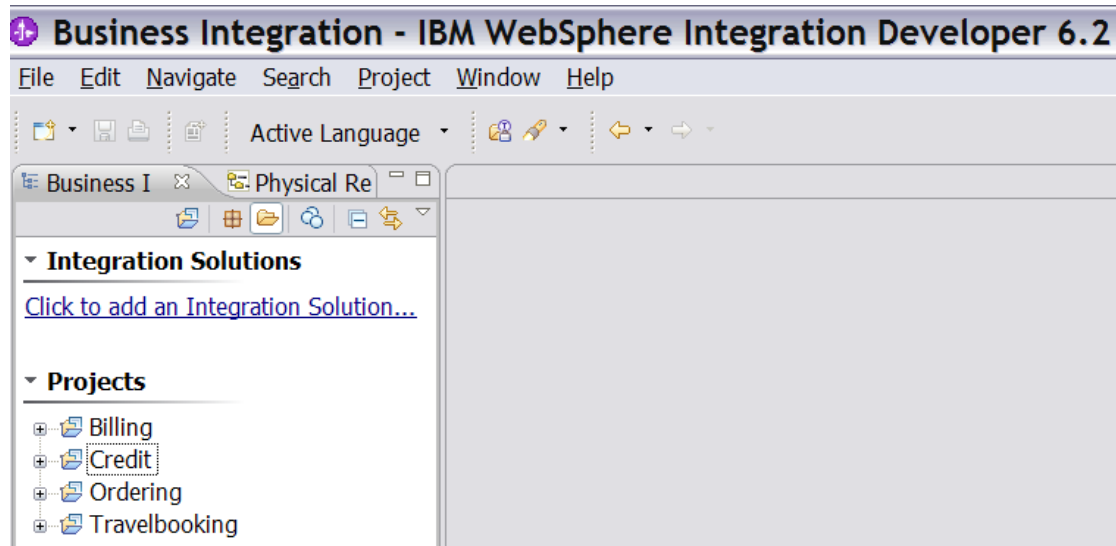


Figure 9: Running FDL2BPEL as WID Migration Wizard (6)

Using as command line tool

This chapter explains how you can invoke FDL2BPEL Conversion from the command line. 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 33). 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 files²³ into WID. Moreover, the command line version has the advantage that it offers additional options to advanced users who want to control more extensively the optimizing of migrated business process models.

Prerequisites

- IBM WebSphere Integration Developer 6.2 (in order to rework or improve the migrated processes)
- IBM WebSphere Process Server, version 6.2 (in order to run FDL2BPEL Conversion from the command line and to deploy and execute the migrated processes)

How to translate WMQWF models into BPC models

To use FDL2BPEL Conversion 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 17).

Running FDL2BPEL Conversion

You can invoke FDL2BPEL Conversion by means of the command file (shell script) "fdl2bpel.bat" (or "fdl2bpel.sh") which you can find in directory:

```
<drive letter>:./<WPS installation directory>/ProcessChoreographer/util
```

FDL2BPEL Conversion has the following general options when running it from the command line:

²³ See "Chapter 3: Understanding the generated files" on page 30.

Table 2: General 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 <u>all</u> files generated by FDL2BPEL Conversion will be written to. Default: Directory location of the input file
-fc	<Boolean>	Treat name conflicts as errors. Because FDL and BPEL have different syntax rules, FDL2BPEL Conversion 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. <ul style="list-style-type: none"> • 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 ²⁴	<Boolean>	Initialize predefined data members. <ul style="list-style-type: none"> • true Create "assign" activities that initialize the predefined data members "_ACTIVITY", "_PROCESS", and "_PROCESS_MODEL". • false (default) Do not initialize the predefined data members
-pn ²⁴	<Boolean>	Do not create predefined data members in BPEL. <ul style="list-style-type: none"> • 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/ "

Assume that your WPS installation directory is "c:\ProcessServer" and that you have an FDL file "CreditRequest.fdl" in directory "c:\MQWF\FDLs". If you intend to store the migration artifacts in directory "c:\BPEL\CreditRequest", then you can run FDL2BPEL Conversion from the command line window as follows (see Figure 10 on page 26):

1. "cd c:\ProcessServer"
2. "fdl2bpel -i c:\MQWF\FDLs\CreditRequest.fdl -od c:\BPEL\CreditRequest"

²⁴ Note that the options "-pi" and "-pn" cannot be used together.

```
C:\ProcessServer\ProcessChoreographer\util>fdl2bpe1 -i C:\MQWF\FDLs\CreditRequest.fdl -od c:\BPEL\CreditRequest
C:\ProcessServer\ProcessChoreographer\util>fdl2bpe1 -i C:\MQWF\FDLs\CreditRequest.fdl -od c:\BPEL\CreditRequest
CWWBM0066I: FDL2BPEL converter version 6.2.
CWWBM0007I: Reading FDL input file "C:\MQWF\FDLs\CreditRequest.fdl".
CWWBM0173I: CreditRequest.fdl(12:18): Converting the structure name "Default Data Structure" in FDL to the struc
CWWBM0056I: CreditRequest.fdl(97:9): The FDL name "CreditRequest" cannot be used because "CreditRequest" is a re
CWWBM0160W: CreditRequest.fdl(97:9): To avoid a name clash, the process name "CreditRequest" in FDL is converted
CWWBM0016I: Writing output file "C:\BPEL\CreditRequest\CreditRequest.xsd".
CWWBM0016I: Writing output file "C:\BPEL\CreditRequest\CreditRequest.wsdl".
CWWBM0039W: CreditRequest.fdl (line number 97 and following): The FDL attribute "valid from" was missing in proc
CWWBM0016I: Writing output file "CreditRequest01_ATASK_PROC.itel".
CWWBM0016I: Writing output file "CreditRequest01_ATASK_ACT.itel".
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 (line number 106 and following): Cannot decide whether the program activity "Accep
CWWBM0021W: CreditRequest.fdl (line number 115 and following): Cannot decide whether the program activity "Asses
CWWBM0021W: CreditRequest.fdl (line number 124 and following): Cannot decide whether the program activity "Colle
CWWBM0021W: CreditRequest.fdl (line number 133 and following): Cannot decide whether the program activity "Rejec
CWWBM0021W: CreditRequest.fdl (line number 142 and following): Cannot decide whether the program activity "Reque
CWWBM0075W: CreditRequest.fdl (line number 97 and following): There is no data connector in process "CreditReque
CWWBM0016I: Writing output file "CreditRequest01_bpel_mon".
CWWBM0016I: Writing output file "C:\BPEL\CreditRequest\CreditRequest01_bpel".
CWWBM0016I: Writing output file "C:\BPEL\CreditRequest\CreditRequest01_bpellex".
CWWBM0016I: Writing output file "CreditRequest01.component".
CWWBM0005I: Exit with return code 2.

C:\ProcessServer\ProcessChoreographer\util>
```

Figure 10: Running FDL2BPEL from the command line

You can find more details on the generated files in "Chapter 3: Understanding the generated files" on page 30.

FDL2BPEL Conversion has the following return codes:

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

In order to make sure that the migrated business process model corresponds to your expectations, you should carefully analyze the displayed messages. Return code 4 often results from errors or missing definitions²⁵ in the FDL file. In this case you may use WMQWF Buildtime to check the correctness of your process models.²⁶

Using optimization options

Besides using the above described general options you can run FDL2BPEL Conversion with optimization options. For more details on optimizing the migrated business model see chapter "Optimizing the migrated business model" on page 96.

²⁵ Check whether you selected option "Export deep" and all required network server definitions when exporting from WMQWF Buildtime (see also "Exporting a WMQWF model from WMQWF Buildtime" on page 17).

²⁶ Import the FDL to WMQWF Buildtime. Open the diagram views of all contained processes and click for each of them on the **Verify** command which you find in the process context menu.

Table 3: FDL2BPEL optimization options²⁷

Command line option	Explanation
-opt	Optimize the generated business process. -opt is equivalent to setting the three options: -mj -es -ev
-mj	Merge Java snippets. ²⁸ -mj is equivalent to setting the two options: -mjp -mjs
-mjp	Serialize multiple parallel Java snippets into on Java Snippet. ²⁸
-mjs	Combine multiple serial Java snippets into on Java Snippet. ²⁸
-es	Remove unnecessary structural elements: ²⁸ <ul style="list-style-type: none"> Remove any "Sequence" that is nested into another "Sequence". Remove any "Parallel Activities" that contain only one entity.
-ev	Allow sharing variables. ²⁸ The optimization algorithm observes the following preconditions (among others): <ul style="list-style-type: none"> common data types no default data values on the inputs and outputs of activities no dynamically assigned activity properties (taken from predefined data members in the input)

Table 4: Additional FDL2BPEL optimization options²⁹

Command line option	Explanation
-id	Tolerate the default data values on the inputs and outputs of activities.
-ip	Tolerate the dynamically assigned activity properties. -ip is equivalent to setting the three options: -ips -ipn -ipp
-ips	Tolerate property "Staff from predefined members".
-ipn	Tolerate property "Notification from predefined members".
-ipp	Tolerate property "Priority from predefined member".

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
3. Enter the desired module name, and click **Finish**.

²⁷ Note that none of the FDL2BPEL optimization options requires a command line argument.

²⁸ This option is ignored if option **-opt** is specified.

²⁹ This options can only be set if the option **"-ev"** is set.

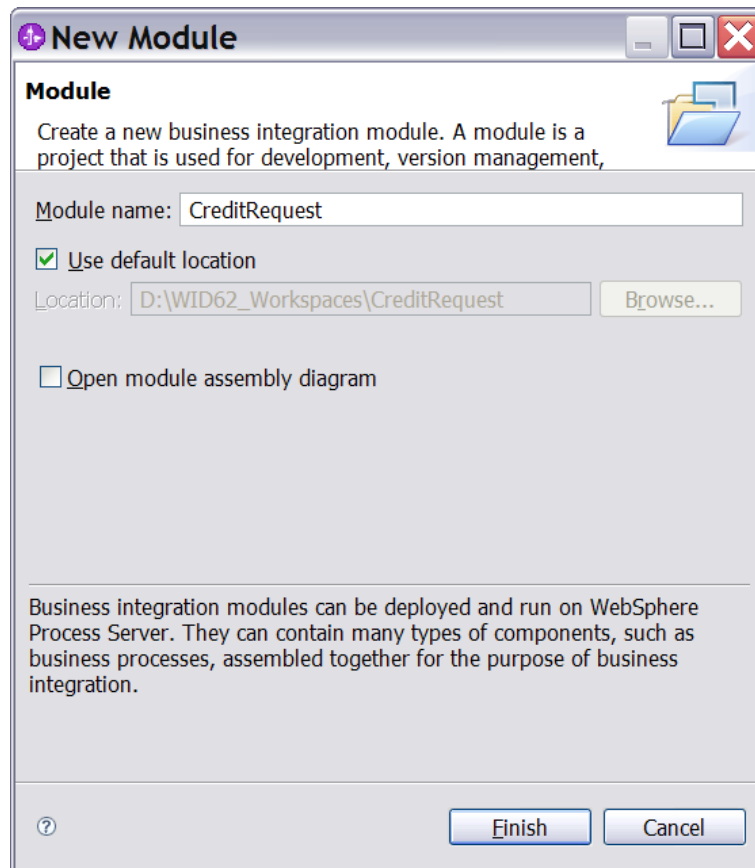


Figure 11: Creating a new business integration module

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

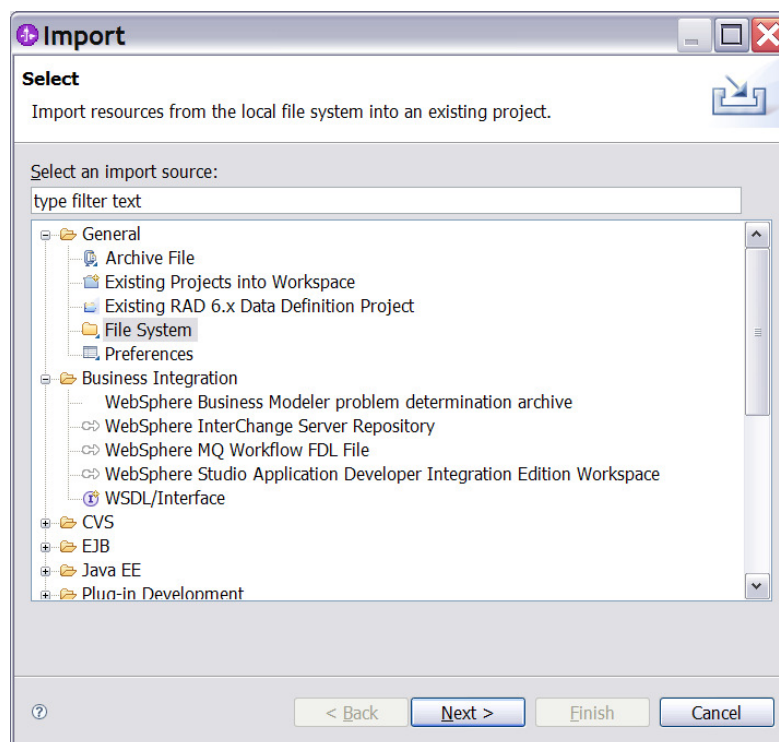


Figure 12: Importing FDL2BPEL generated files (1)

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

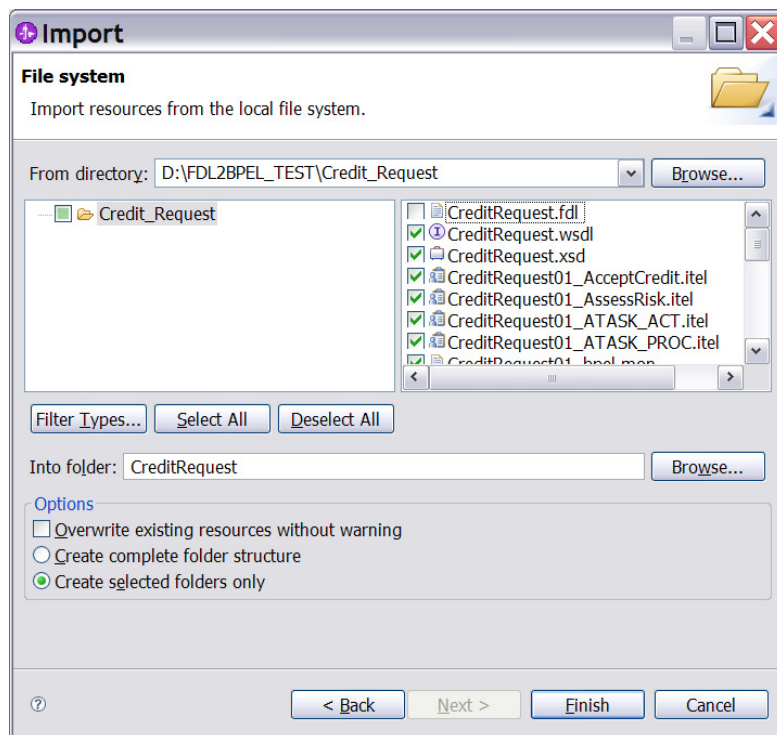
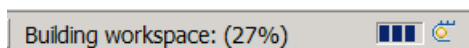


Figure 13: Importing FDL2BPEL generated files (2)

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



Chapter 3: Understanding the generated files

Introduction

FDL2BPEL Conversion generates the following file types:

Table 5: 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 ³⁰	<FDL process name>.bpelex	Partial XML description of the graphical layout of a single business process model
TEL	<FDL process name>_ATASK_PROC.itel	XML description of a task that is assigned to (a) person(s) who administer(s) the business process
TEL	<FDL process name>_ATASK_ACT.itel	XML description of a task that is assigned to (a) person(s) who administer(s) the activities of the business process
TEL	< FDL activity name>.itel	XML description of a task that is assigned to (a) person(s) who participate(s) in the execution of an activity of the business process
TEL	< FDL activity name>_ATASK_ACT.itel	XML description of a task that is assigned to (a) person(s) who administer(s) an activity of the business process
COMPONENT	<FDL process name>.component	XML description of an SCA process component
IMPORT	<system name>_<server name>.import	XML description of an SCA MQ JMS Import (required for a UPES invocation)
JAVA	<FDL data structure name>_UPES_OUT_MSG_DataBindingImpl.java	Data binding for SCA MQ JMS Import (required for a UPES invocation)
MON	<FDL process name>_bpel.mon	Definitions for monitoring events

The XSD file

For each FDL file, FDL2BPEL Conversion 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")³¹ or uncheck the Migration Wizard option "Create predefined data members"³².

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

³⁰ The generated BPELEX files contain the required XML definitions that control the status "collapsed" of the BPEL structured activities.

³¹ See "Running FDL2BPEL Conversion" on page 24.

³² See "Using as WID Migration Wizard" on page 18.

The WSDL file

A WSDL (Web Services Definition Language) file describes Web services in a common XML grammar. FDL2BPEL Conversion 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, FDL2BPEL Conversion 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" (BPEL4WS³³). 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.³⁴ In particular, the file contains control information about the initial "collapsed" status of BPEL structured activities.

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 6.2, 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

FDL2BPEL Conversion 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"³⁵, 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.

³³ In this documentation BPEL4WS Version 1.1 with IBM extension elements is referred to as "BPEL".

³⁴ WID automatically adds other layout definitions to the generated BPELEX file(s).

³⁵ See "Activity classification rules" on page 57.

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:

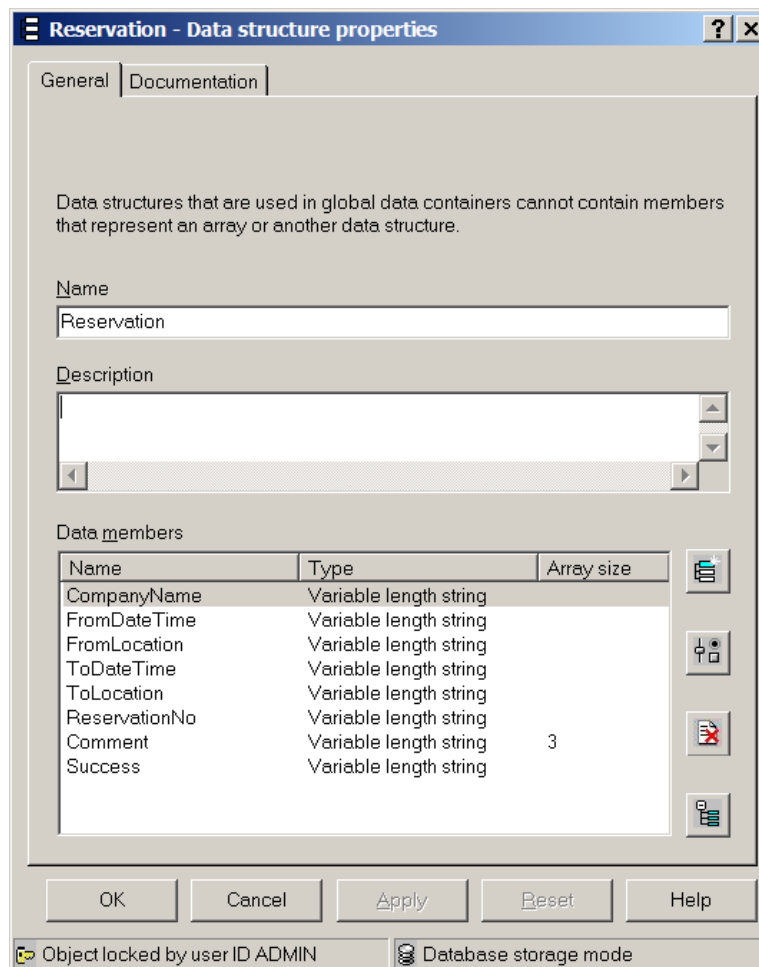


Figure 14: Sample FDL data structure

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'

```

FDL2BPEL Conversion 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 6: Mapping basic FDL data types

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

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.³⁶

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

³⁶ See "Data arrays" on page 135.

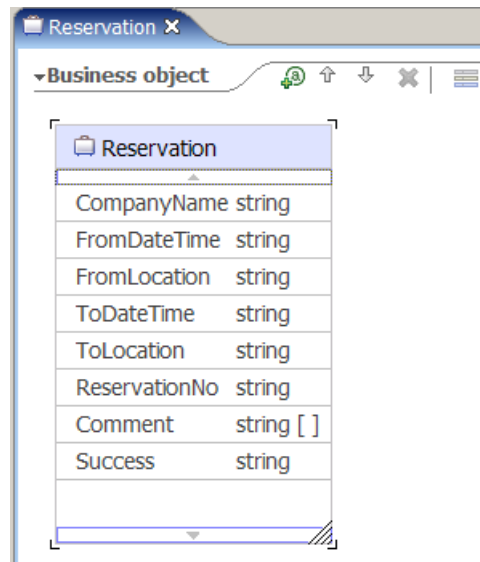


Figure 15: WID Business Object Editor

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

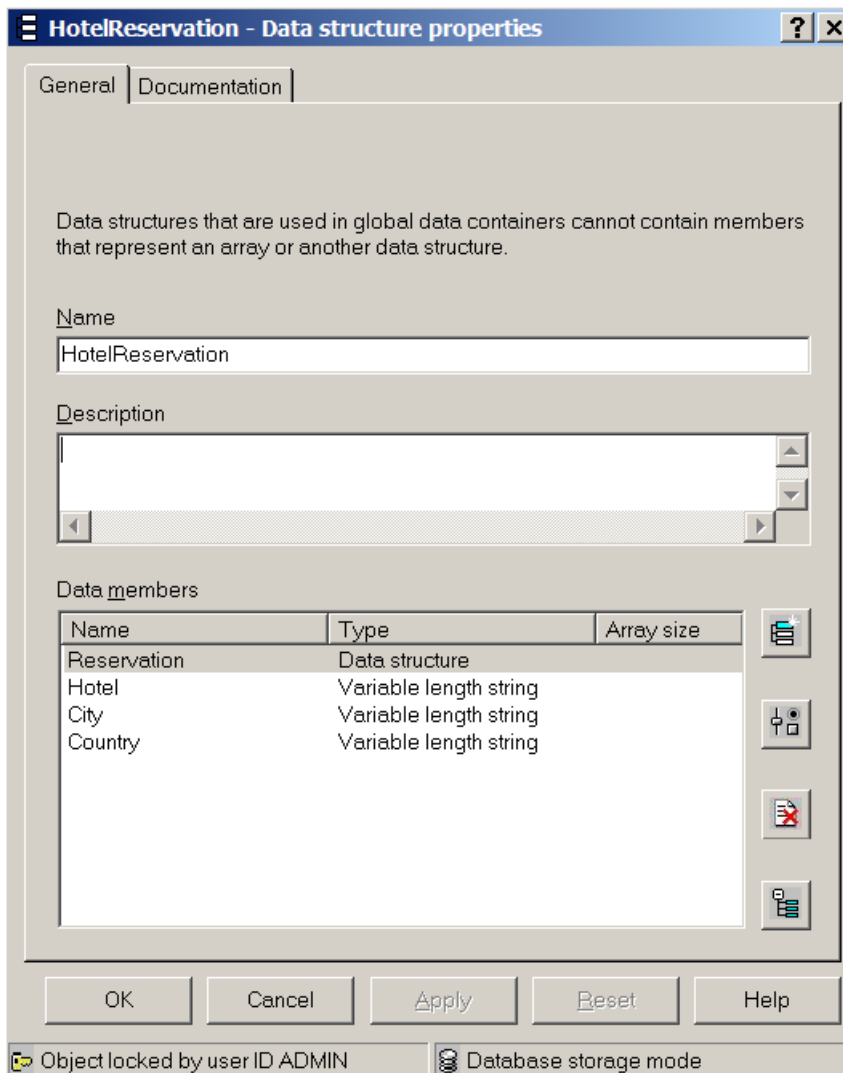


Figure 16: Hierarchical FDL data structure

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 FDL2BPEL Conversion:

```
<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:

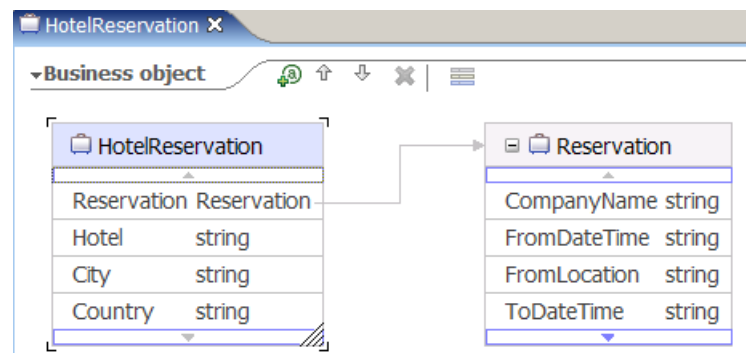


Figure 17: Hierarchically structured Business Object

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 18: 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 FDL2BPEL Conversion.

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:

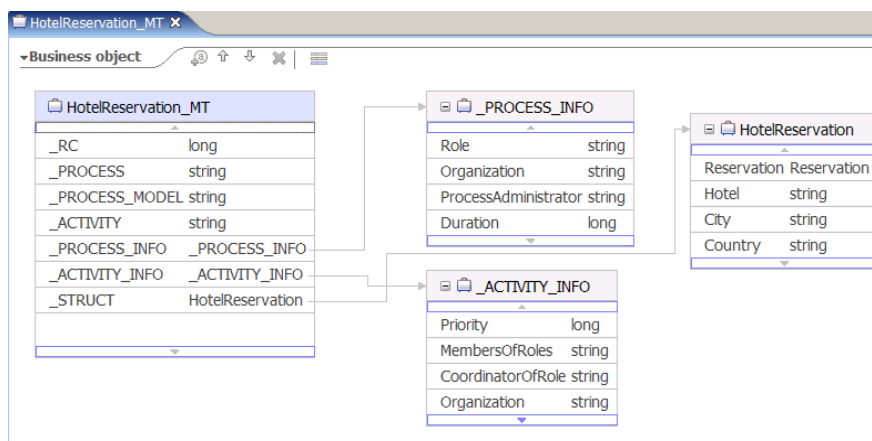


Figure 19: Mapping FDL data container to Business Object

You can suppress the generation of predefined data members by using the command line option **-pn** ("do not generate predefined data members")³⁷ or by unchecking the Migration Wizard option "Create predefined data members"³⁸. 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"). Using option **-pn** requires that the FDL file does not contain any references to predefined data members. Here are examples for such references that prevent using option **-pn**:

- Data mappings between predefined data members
- Property "Staff from predefined members"
- Property "Notification from predefined members"
- Property "Priority from predefined members"³⁹

³⁷ See "Running FDL2BPEL Conversion" on page 24.

³⁸ See "Using as WID Migration Wizard" on page 18.

³⁹ Note that property "Priority from predefined members" is selected by default in WMQWF Buildtime!

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"). FDL2BPEL Conversion 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 a corresponding "UPES Web service" in a WPS environment.⁴⁰ 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">41
  <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>
```

⁴⁰ **Warning:** The UPES input/output messages described in this chapter represent special data structures which are required by the MQ JMS binding (see "SCA Imports" on page 89) in order to perform the automatic translation of WPS business objects into WMQWF XML messages (as described in the WMQWF programming guide) and vice versa at runtime. That way, the MQ JMS binding supports reusing existing UPES applications in a WPS environment.

If you want to write a new UPES application for a WPS business model using the MQ JMS binding, you have to program it according to the MQ Workflow XML message format which is different from the one outlined in this documentation!

⁴¹ Note that 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. migration hint "XML message types" on page 143).

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:

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>
```

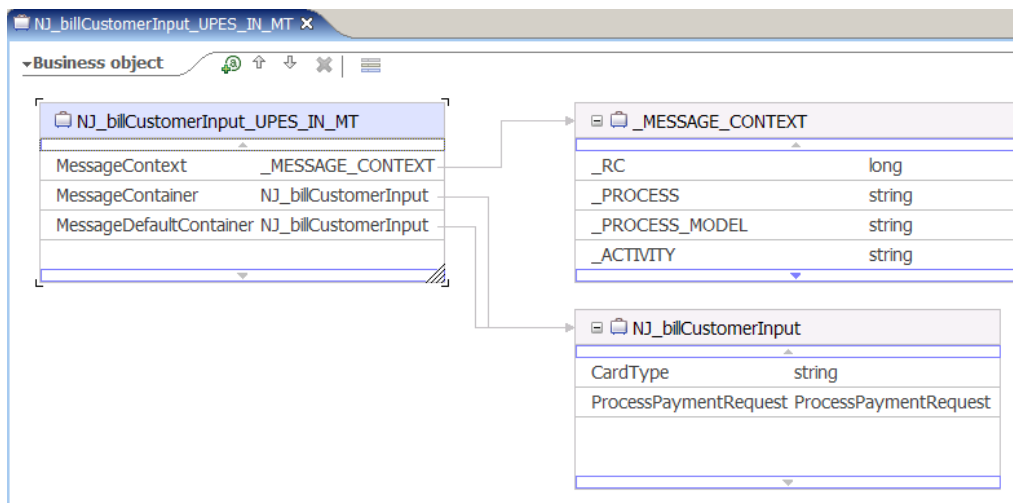


Figure 20: UPES input message as Business Object

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>
```

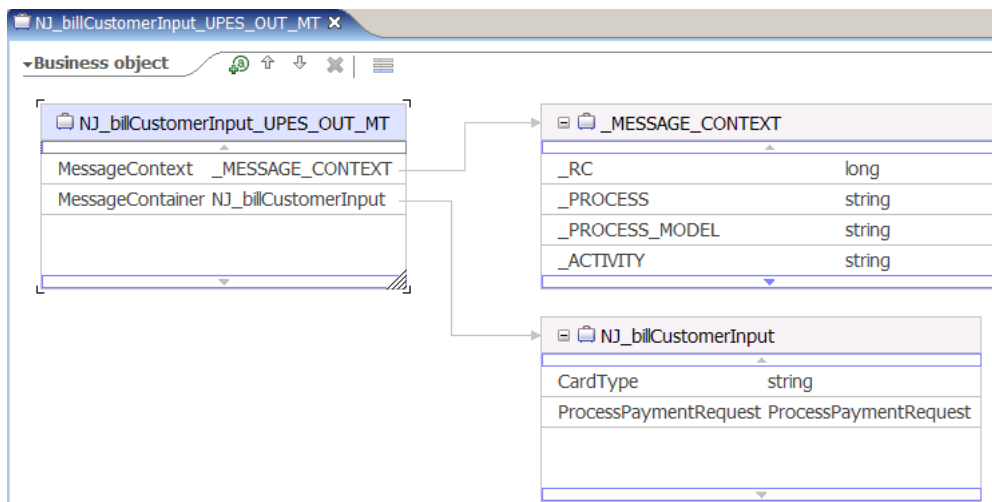


Figure 21: UPES output message as Business Object

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, FDL2BPEL Conversion creates a partner link definition. For example⁴²:

```
<!-- 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.

A second partner link type is added, if the "Nice_Journey" process has a subprocess "NJ_BookCar":

```
<!-- 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 FDL2BPEL Conversion 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>
```

⁴² 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.

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.

FDL2BPEL Conversion creates messages that represent FDL container data⁴³ 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:

```
<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>
```

FDL2BPEL Conversion 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 69):

```
<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

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

⁴³ See "Mapping an FDL data container to an XML schema definition" on page 37.

```

<!-- 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:

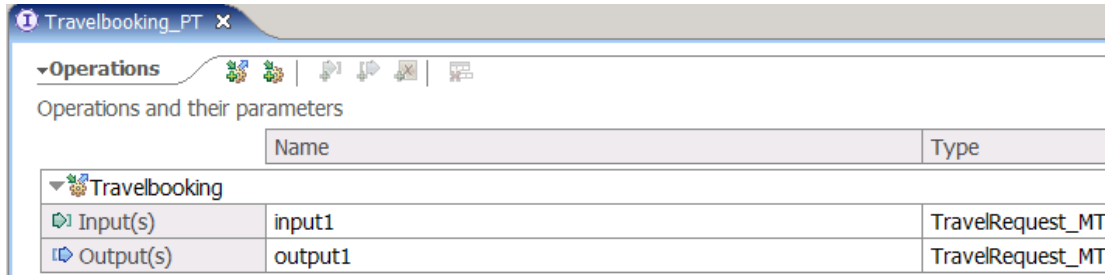


Figure 22: Process interface

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.

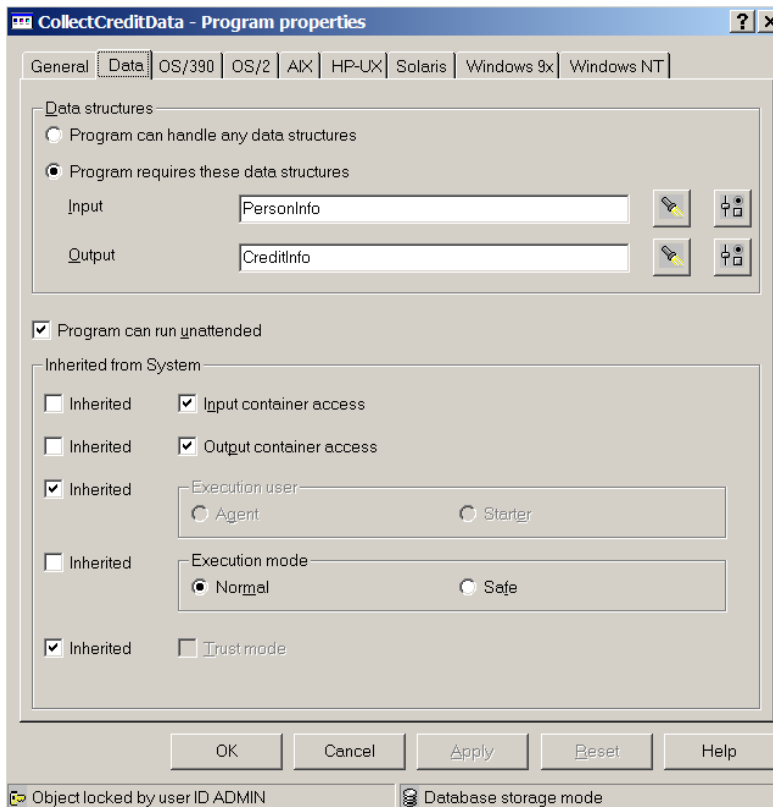


Figure 23: Program requires these data structures

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:

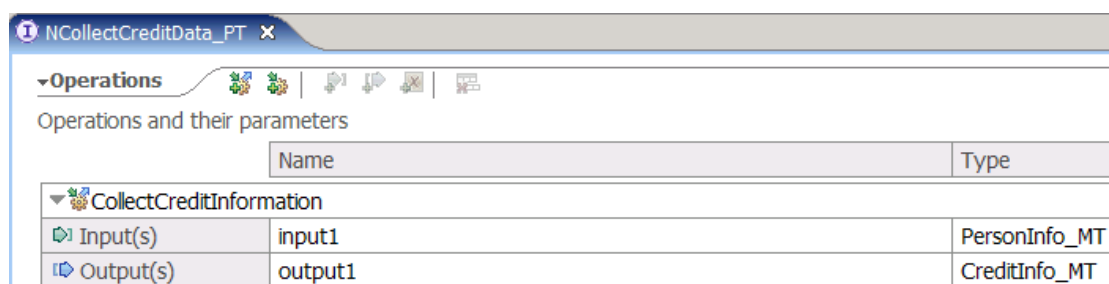


Figure 24: Interface with single operation

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".

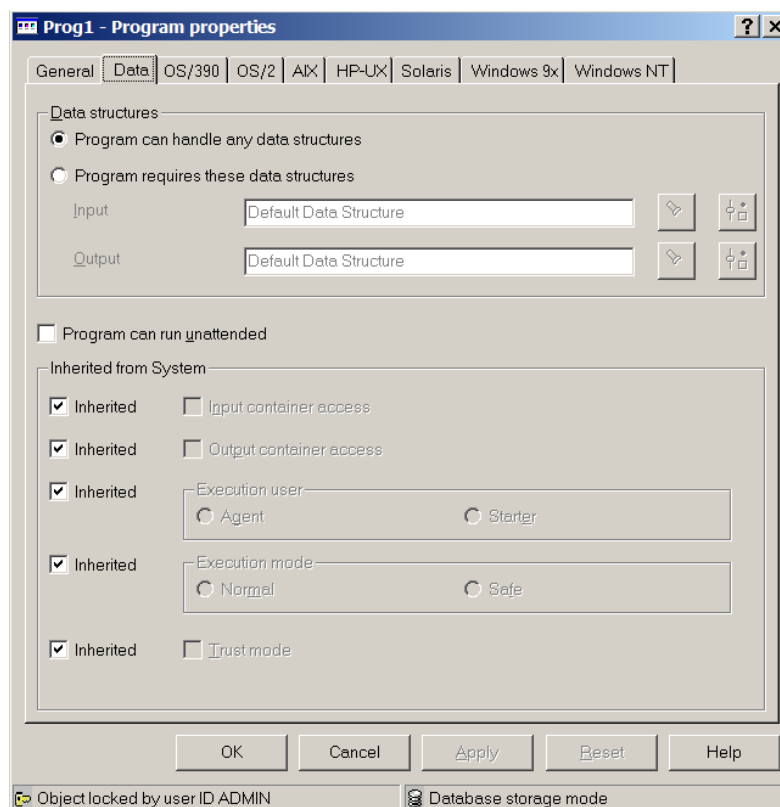


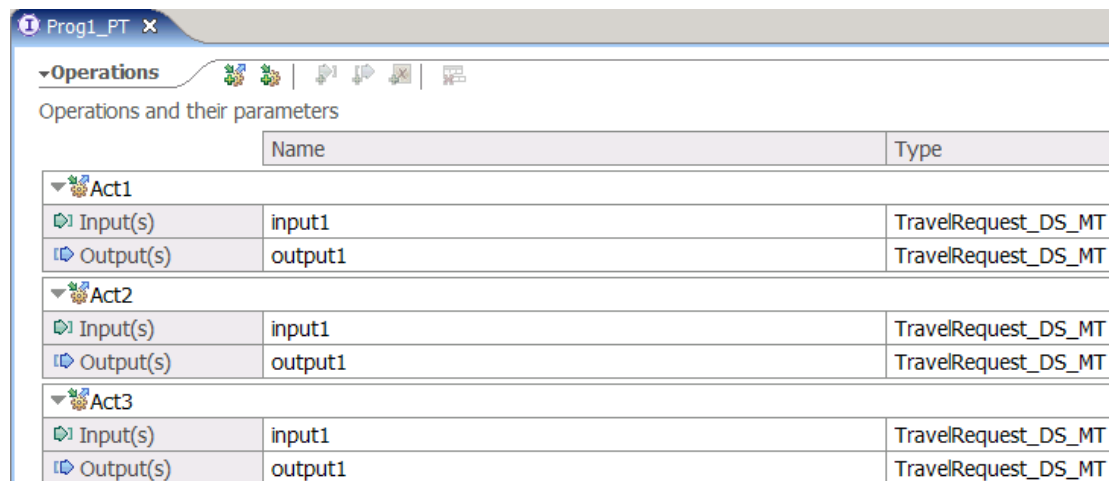
Figure 25: Program can handle any data structures

For instance, if activities "Act1", "Act2", "Act3" specify program "Prog1", FDL2BPEL Conversion 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:



Operations and their parameters		
	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 26: 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:

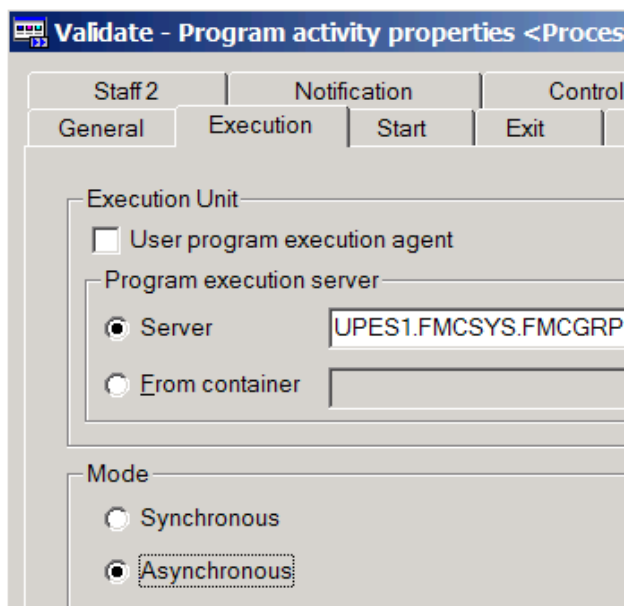


Figure 27: Asynchronous execution of UPES

Mapping FDL SERVER to a port type for a UPES

As described in the section "Mapping the data exchanged with a UPES" on page 40, 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:

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 28: 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", FDL2BPEL Conversion 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, FDL2BPEL Conversion creates a separate file with the extension ".bpe1". 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:

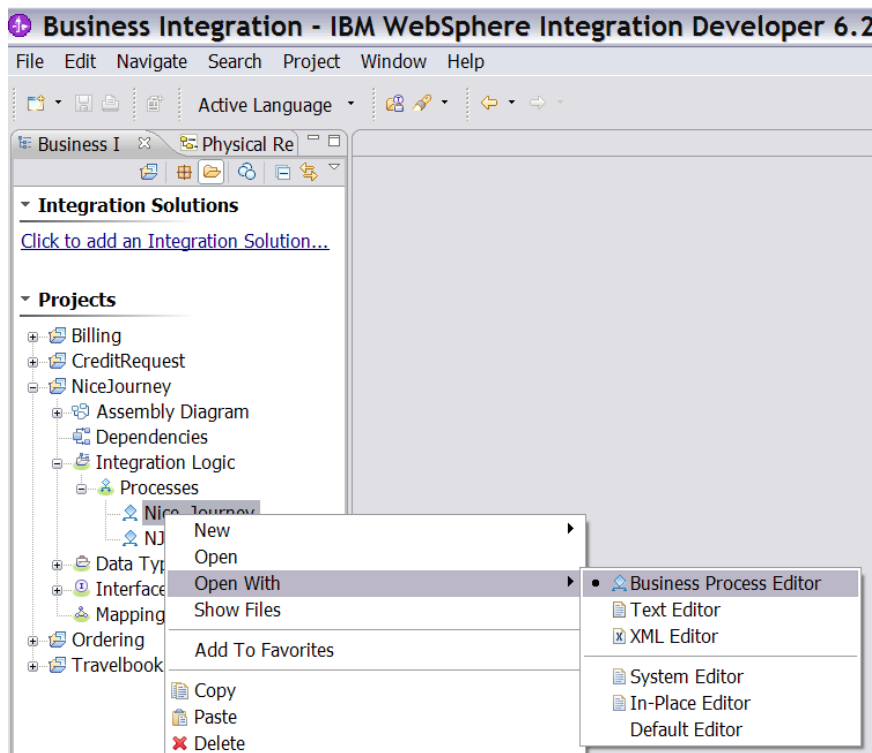


Figure 29: Opening the business process editor

2. Select the Description page of the Properties view:

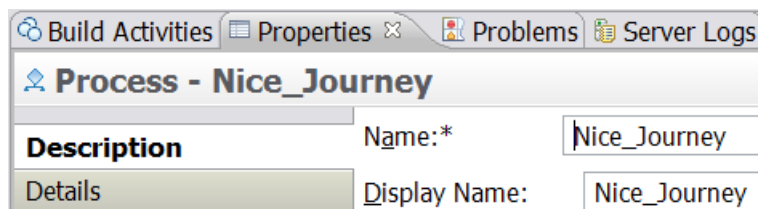


Figure 30: Mapping process properties (1)

Further examples of process properties are:⁴⁴

⁴⁴ 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:

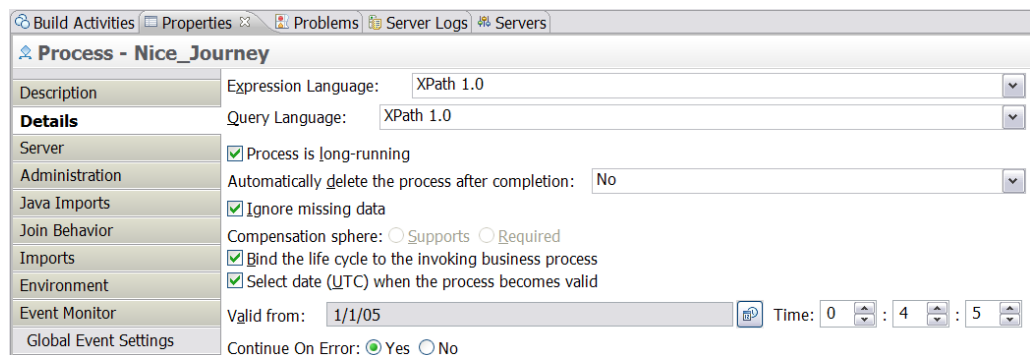


Figure 31: Mapping process properties (2)

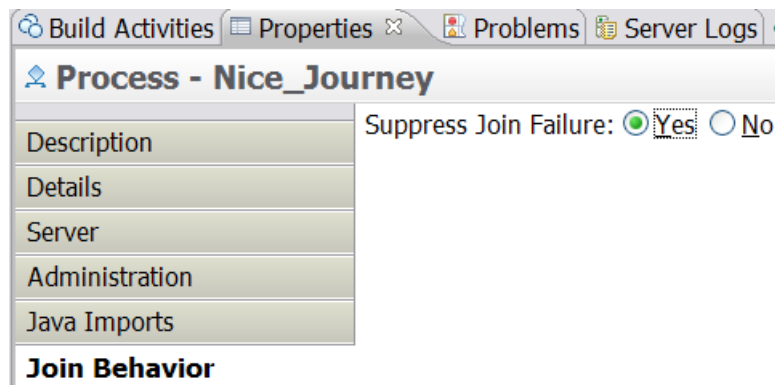


Figure 32: Mapping process properties (3)

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", FDL2BPEL Conversion 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="yes" ... >
```

WID business process editor:

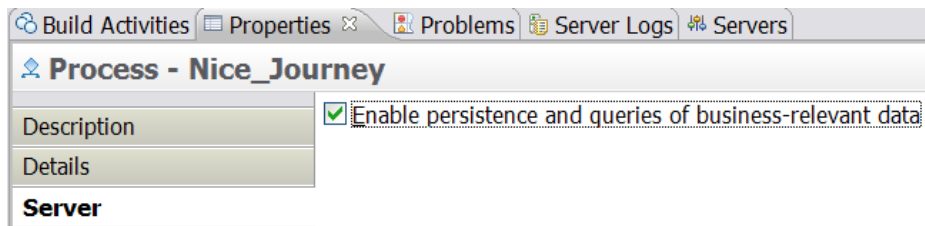


Figure 33: Mapping process properties (4)

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.

FDL2BPEL Conversion 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**, which means that the life cycle of the process is bound to any invoking parent process (see Figure 31 on page 49).⁴⁵

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:

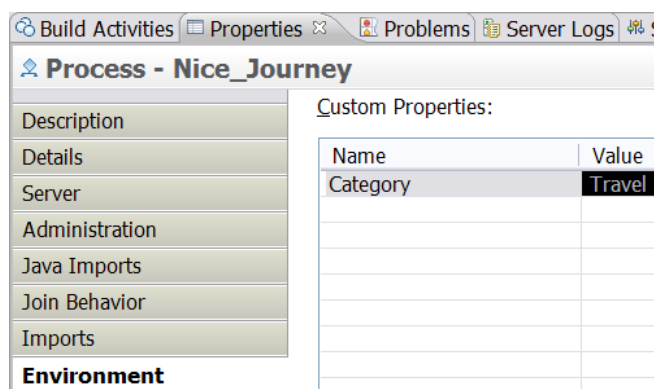


Figure 34: Process Category

⁴⁵ See migration hint "Process autonomy modes" on page 145.

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:

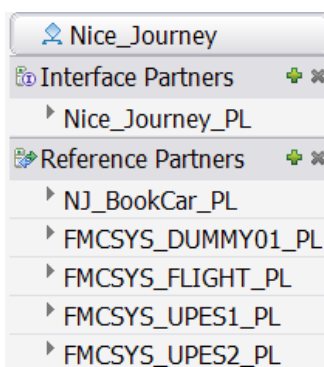


Figure 35: Partner links

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, FDL2BPEL Conversion 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>
  <!-- Default input variable of process "SimpleProcessWDF" -->
  <bpws:variable name="SimpleProcessWDF_DEFAULT_IN" type="mqwf:TravelRequest_DS_MT"/>
  <!-- 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:

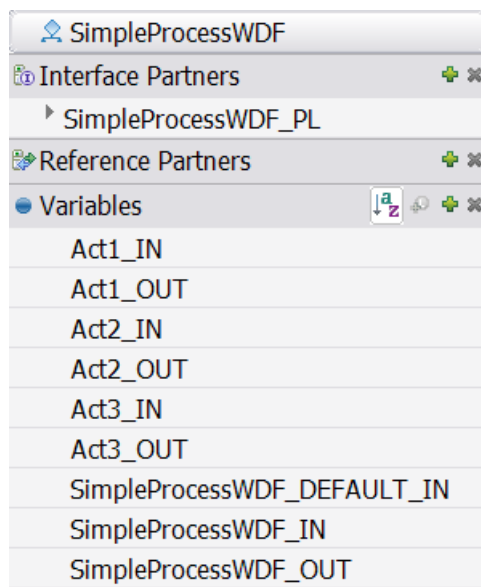


Figure 36: Mapping FDL data containers to BPEL variables

Queryable 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 FDL2BPEL Conversion 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>

```

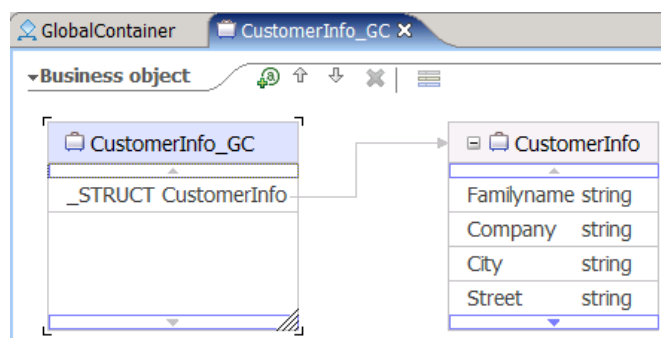
WID business object editor:

Figure 37: Mapping Global Container to business object

WID Properties view of BPEL variable "GLOBAL_CONTAINER":

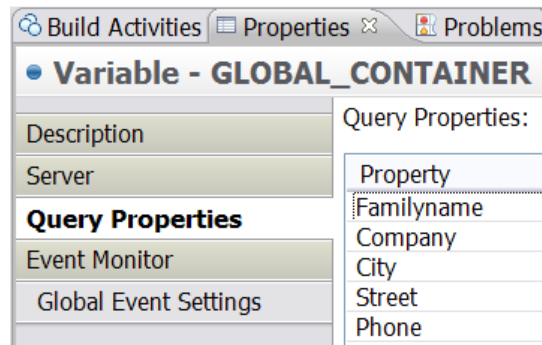


Figure 38: Mapping FDL data structure to BPEL query properties

Mapping the FDL workflow to BPEL

The following BPEL activity framework outlines the general structure that is used by FDL2BPEL Conversion to represent every FDL workflow⁴⁶:

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>
```

BPEL business process editor:

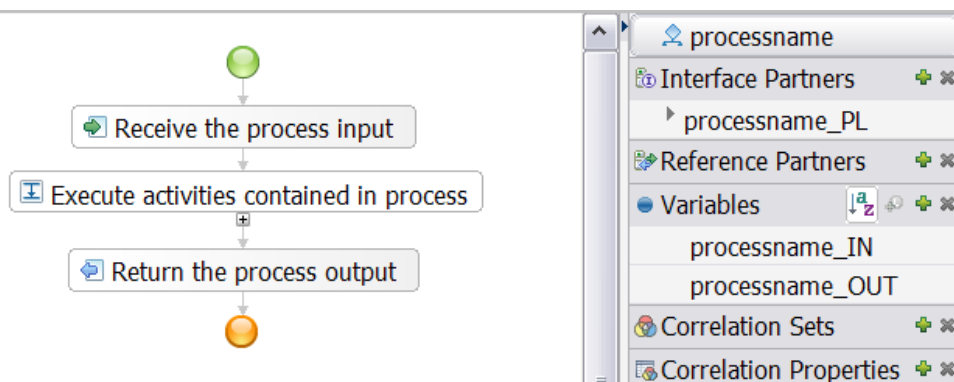


Figure 39: Mapping the workflow structure (1)

⁴⁶ See also "Optimizing BPEL process models" on page 96

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.

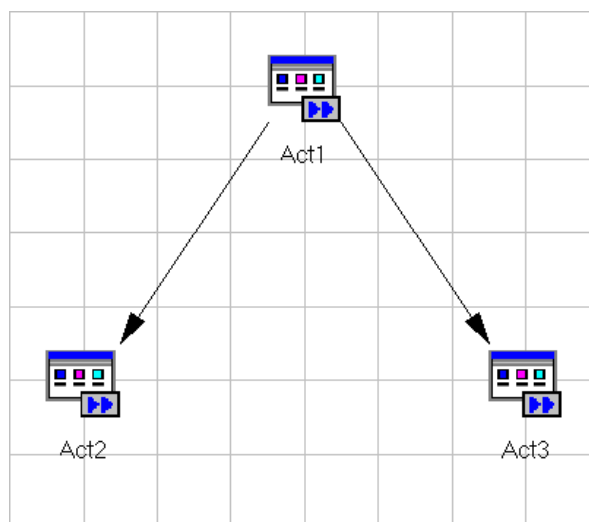


Figure 40: Mapping the workflow structure (2)

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:

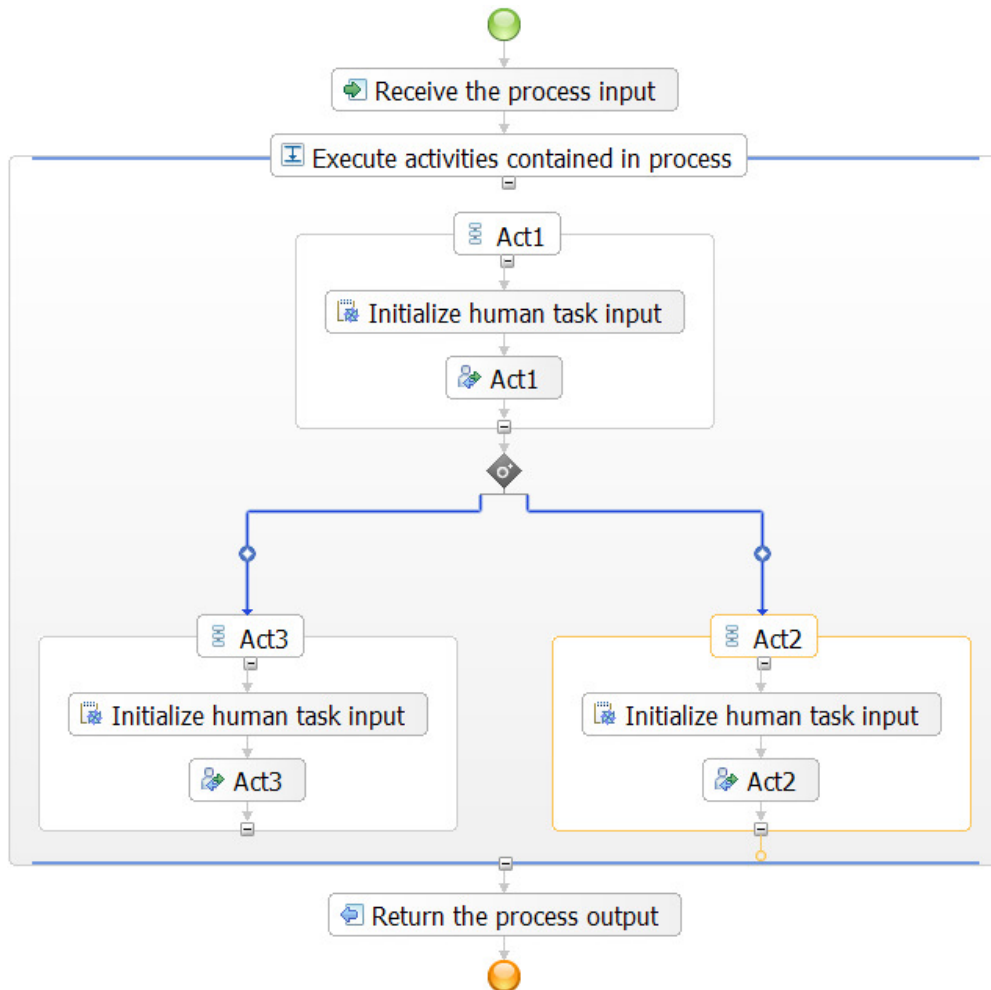


Figure 41: Mapping the workflow structure (3)

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

FDL2BPEL Conversion 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 migration hint "FDL program activity" on page 146). 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 FDL2BPEL Conversion 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 7: Activity classification rules

	Activity type ⁴⁷	Mapping rule
1.	"Empty activity"	A program activity maps to an "empty activity" if <ol style="list-style-type: none"> the activity implementation is a program with the name "FMCINTERNALNOOP", the execution mode is asynchronous⁴⁸ (see the properties notebook page "Execution"), and 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 <ol style="list-style-type: none"> its start type is "manual" (see the properties notebook page "Start"), and 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":

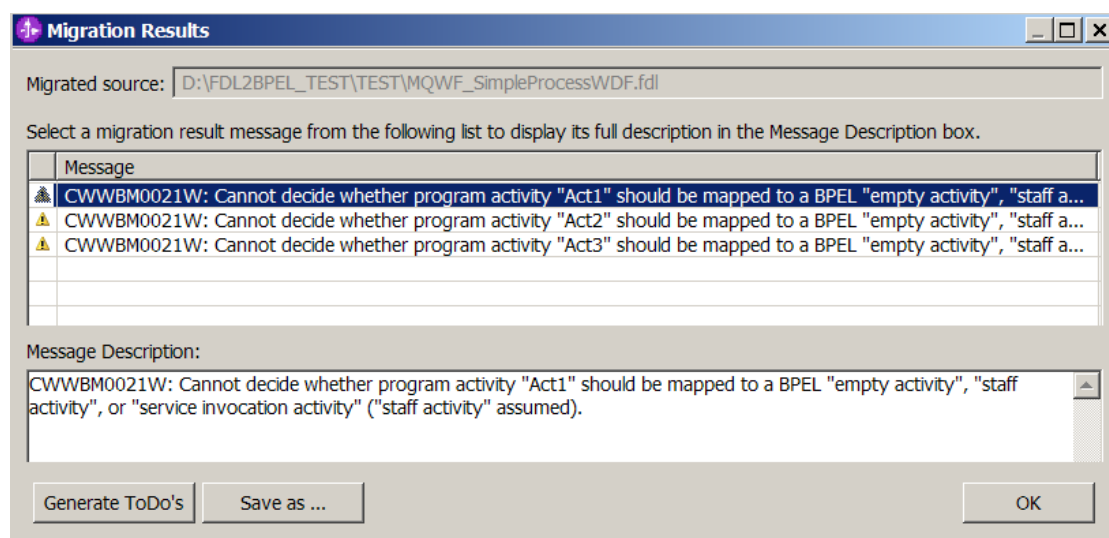


Figure 42: Assumed "staff activity"

⁴⁷ 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.

⁴⁸ 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":

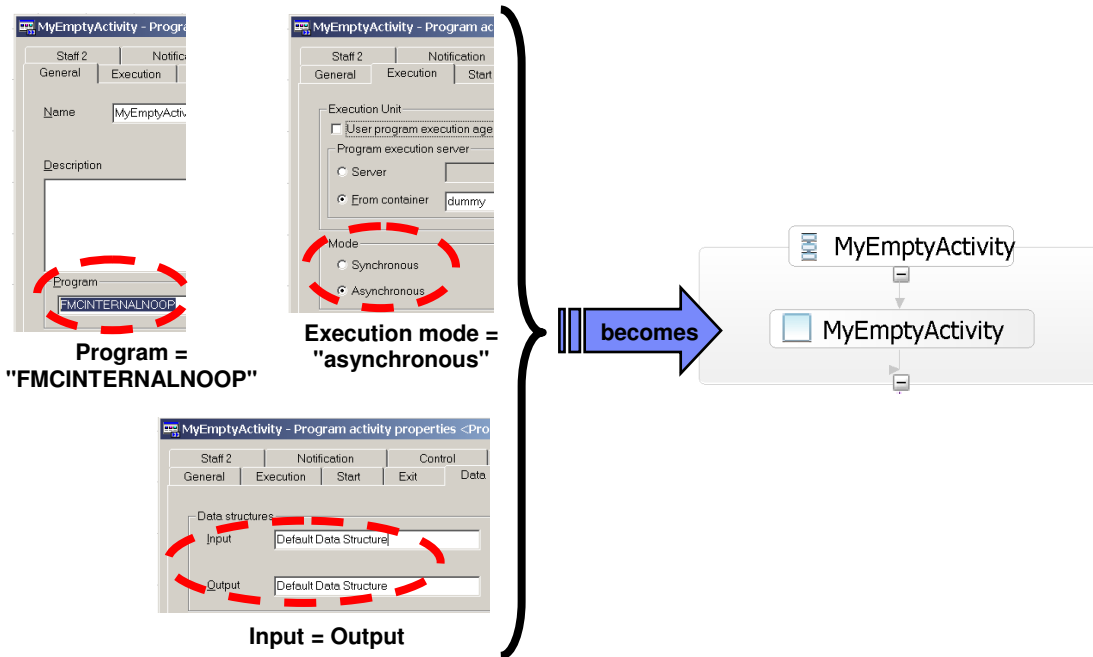


Figure 43: Activity classification rule: "empty activity"

BPEL source code:

```
<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":⁴⁹

⁴⁹ See also chapter "WMQWF UPES migration" on page 95.

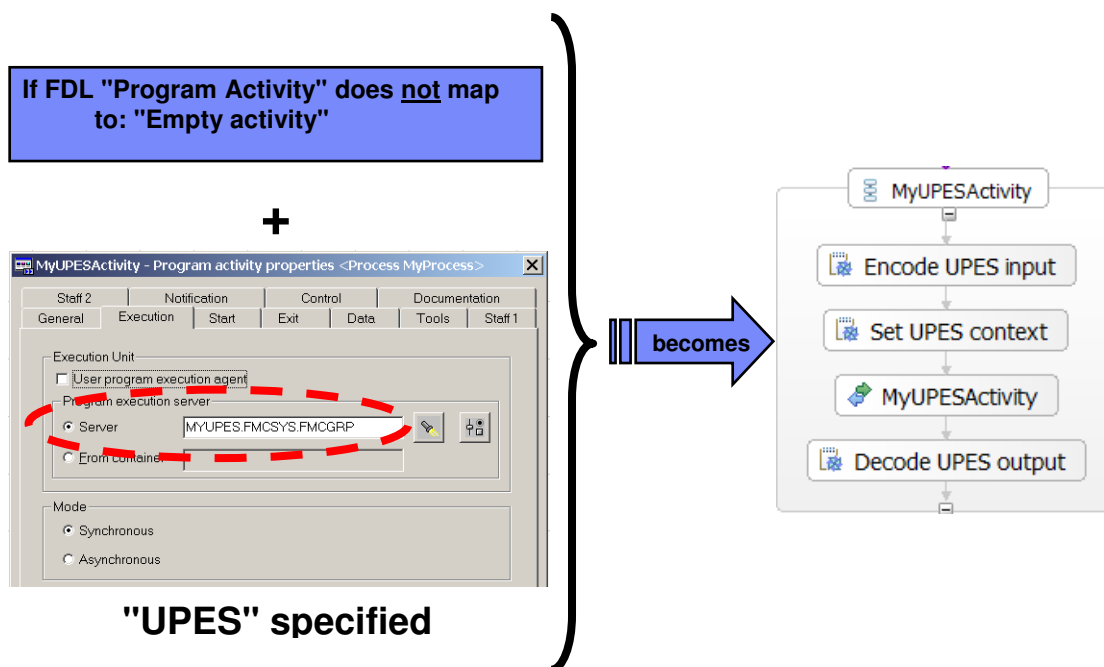


Figure 44: Activity classification rule: "service invocation activity"

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 40, 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 73, 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).

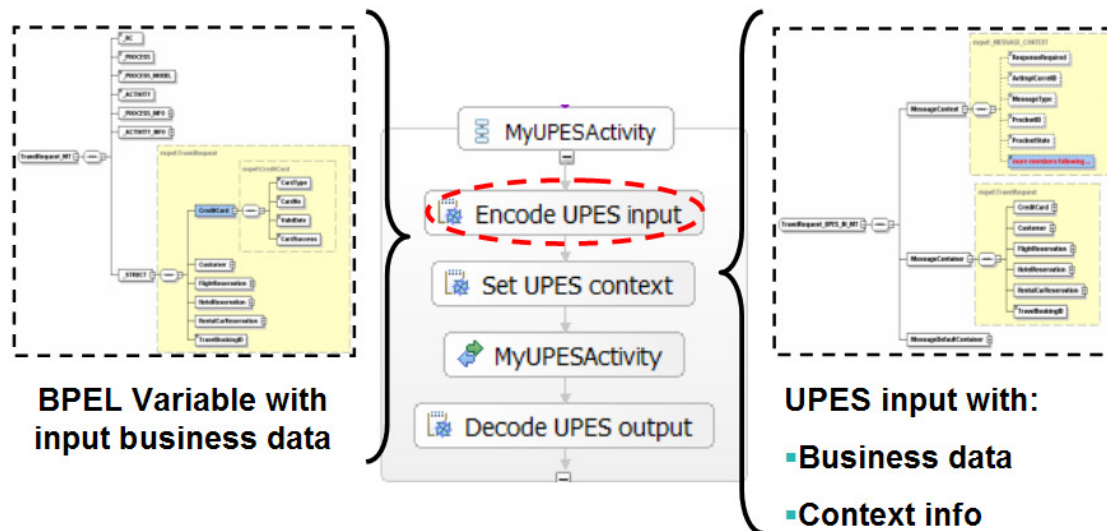


Figure 45: Encoding the UPES input

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":

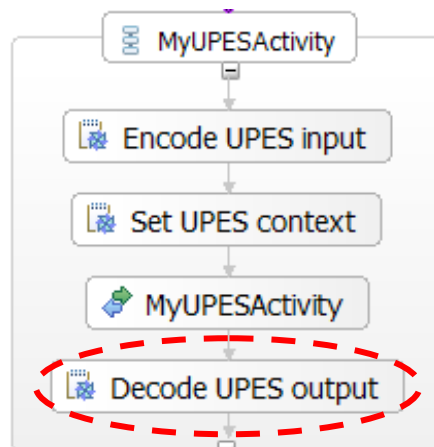


Figure 46: Decoding the UPES output

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.

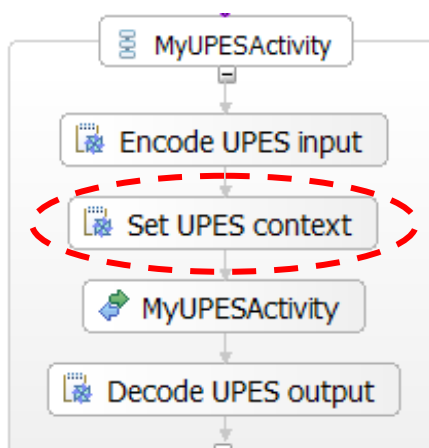


Figure 47: Setting the UPES context

The snippet "Set UPES context" (see Figure 47) adds context data to the UPES message and initializes the following elements of the XML message that the UPES will receive:

Table 8: 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 73.

Staff activity

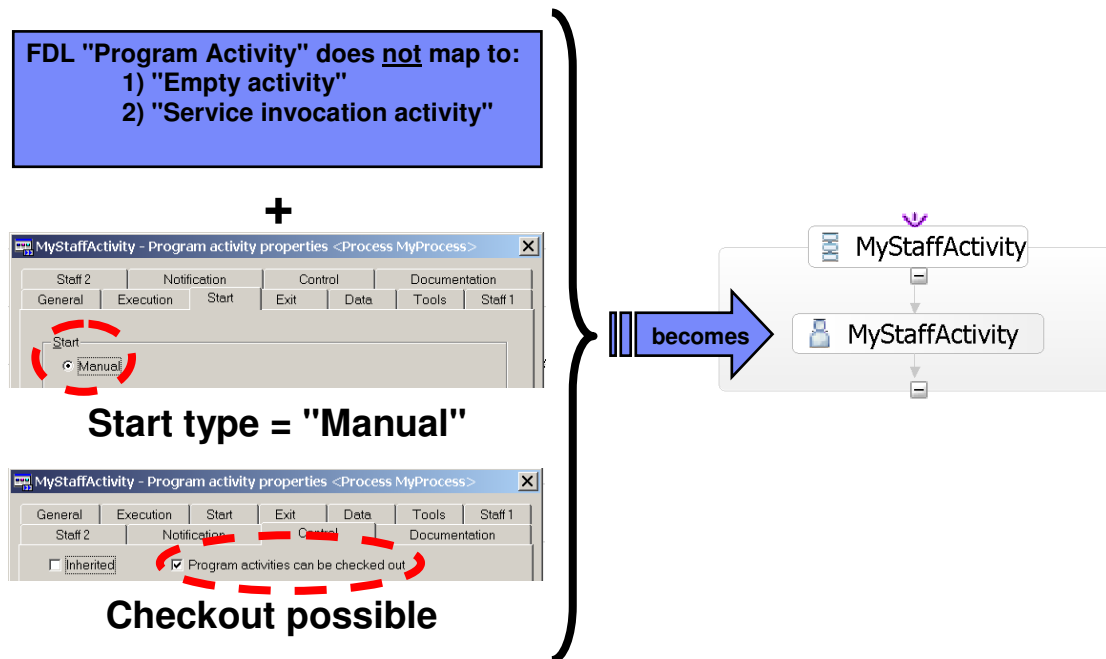


Figure 48: Classification rule "staff activity"

BPEL source code:

```

<!-- Invoke program activity "MyStaffActivity" -->
<bpws:invoke name="MyStaffActivity"
  operation="MyStaffActivity"
  portType="wsdl: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 80):

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>

```

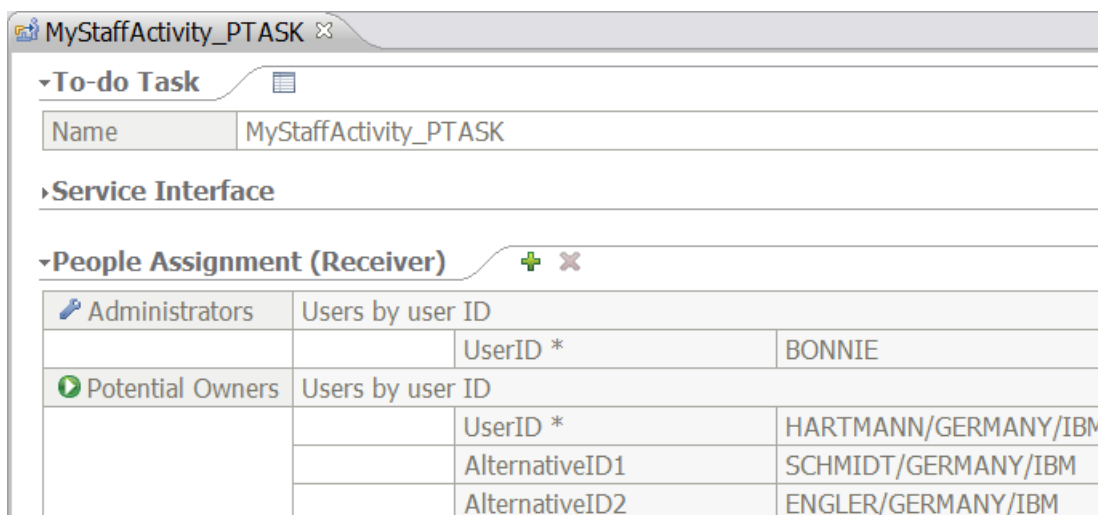


Figure 49: WID human task editor

Activity property "business relevant"

FDL2BPEL Conversion enables the "business relevant" property of the generated BPEL "invoke" activity, if the FDL activity has property "AUDIT_FILTER_DB" or property "AUDIT_FILTER_MQ" or if an audit trail filter is specified with values "condensed" or "full" on page "Control" of the process properties notebook. Otherwise, the "business relevant" property is disabled.

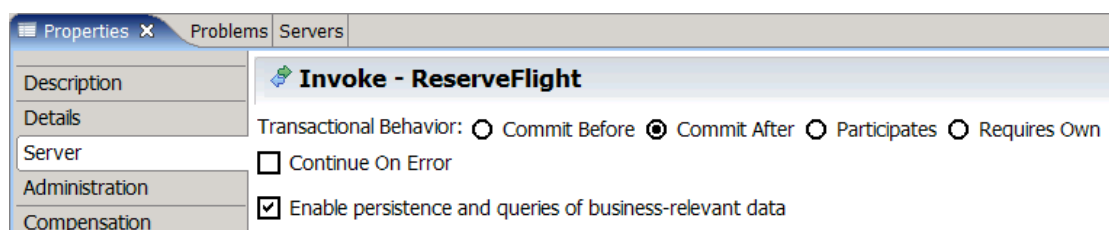


Figure 50: Mapping activity property "business relevant"

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:

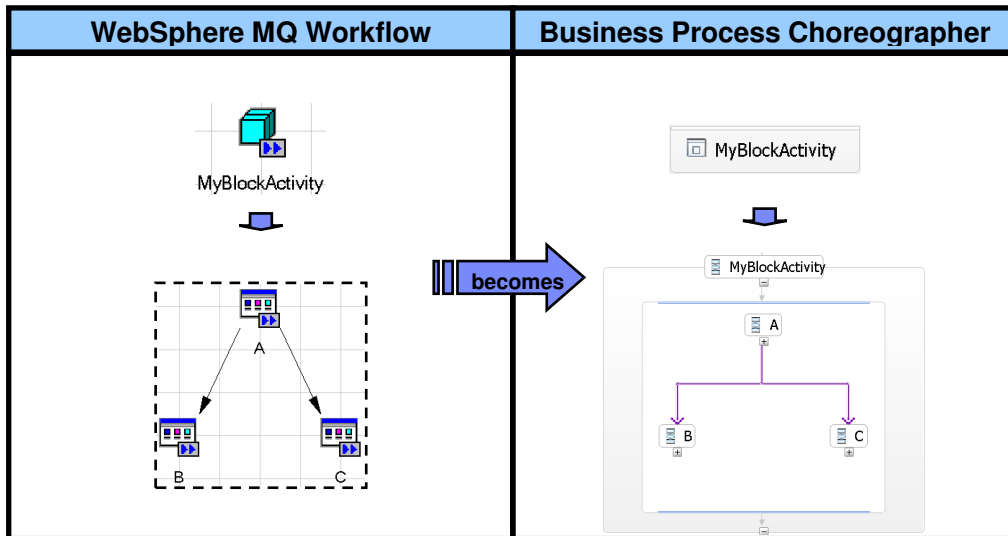


Figure 51: Mapping an FDL block to BPEL

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 69 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:

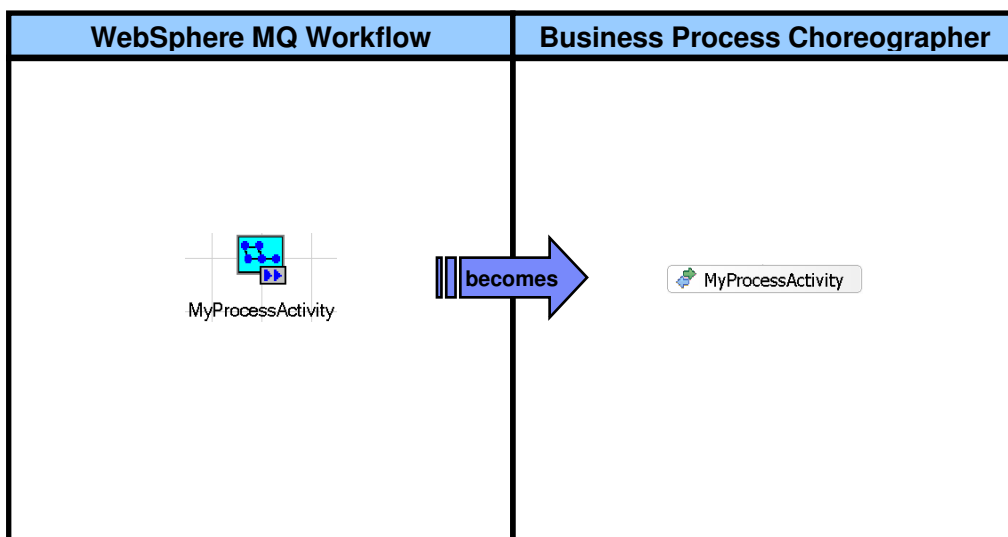


Figure 52: Mapping an FDL subprocess call to BPEL

The following figure shows the invoke implementation properties sheet:

	Name	Type	Variable
Input(s)	input1	TravelRequest_MT	Process_IN
Output(s)	output1	TravelRequest_MT	Process_OUT

Figure 53: 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):

Figure 54: 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:

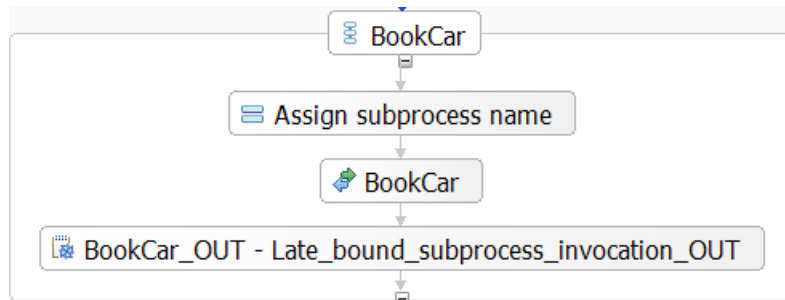


Figure 55: Late-bound subprocess invocation

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:copy>
</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_OUT" name="output1"/>
  </wpc:output>
</bpws:invoke>
...

```

WID BPEL editor with properties view of assign activity:

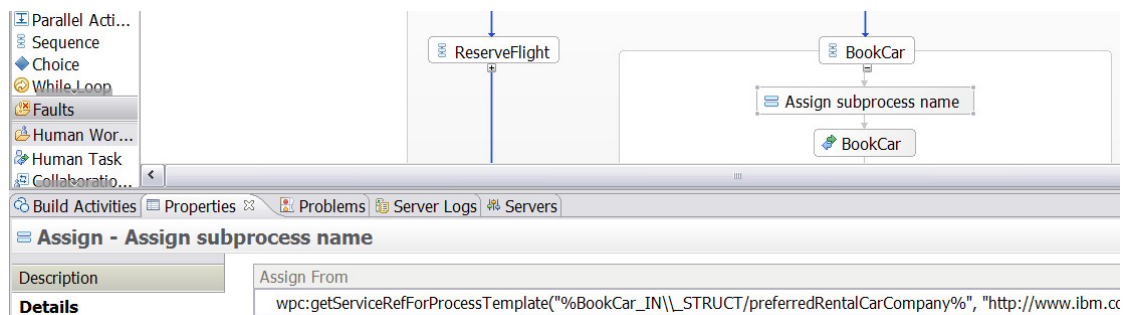


Figure 56: Assigning the subprocess name

Mapping WMQWF control flow to BPEL

In "Mapping the FDL workflow to BPEL" on page 54, 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 FDL2BPEL Conversion.

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:

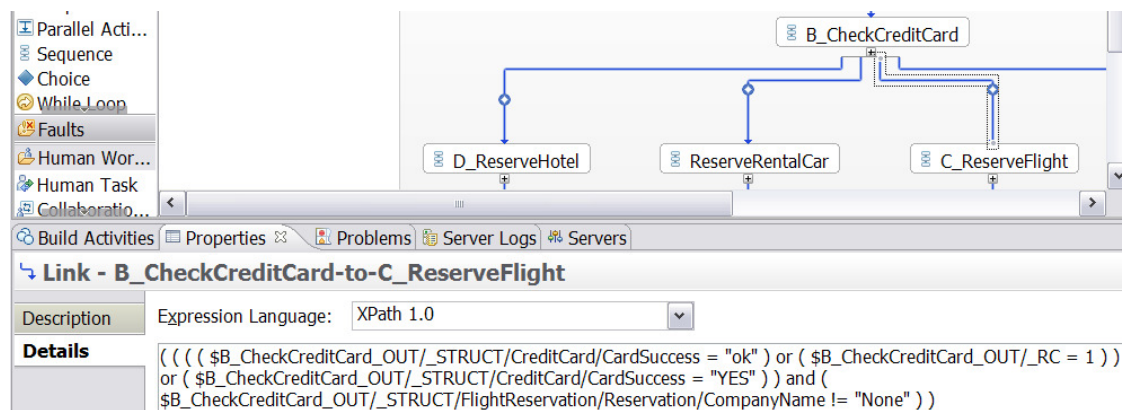
```
CONTROL
FROM 'B_CheckCreditCard' TO 'C_ReserveFlight'
WHEN "(CreditCard.CardSuccess= ""ok""
      OR _RC= 1
      OR CreditCard.CardSuccess= ""YES"" )
      AND FlightReservation.Reservation.CompanyName<> ""None"" "
```

BPEL source code:

```

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

```

WID BPEL editor with properties view of link connector:**Figure 57: 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>
  ...

```

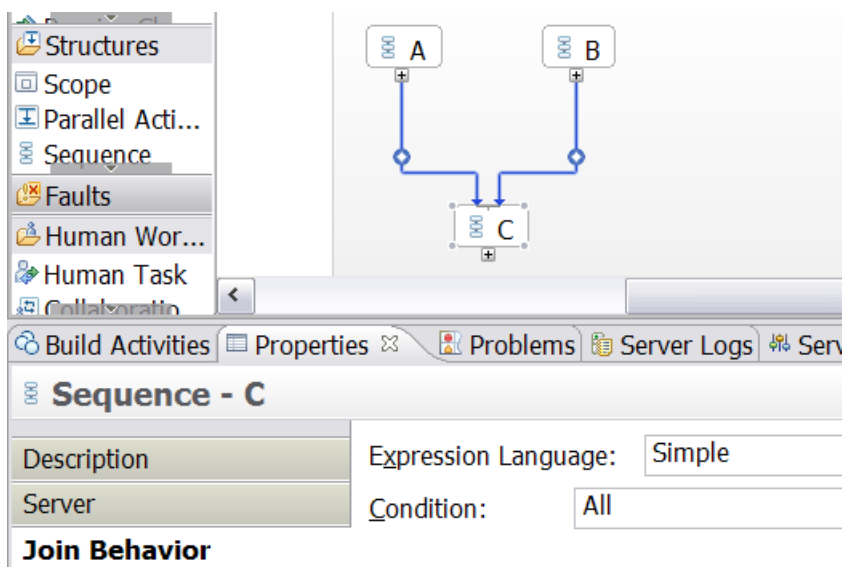
WID BPEL editor with details view of start condition:

Figure 58: 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
    activity
  </while>

```

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

Table 9: Mapping an exit condition expression to BPEL

FDL <i>exit condition</i>	BPEL <i>while condition</i>
...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.

FDL2BPEL Conversion 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. FDL2BPEL Conversion 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( ( $BlockEnterandApprove_OUT/_STRUCT/OrderData/OrderStatus != "postponed" ) )
    or not( $BlockEnterandApprove_ST/ActivityStatus = "started" )
  </bpws:condition>
  ...
</bpws:while>
```

WID BPEL editor:

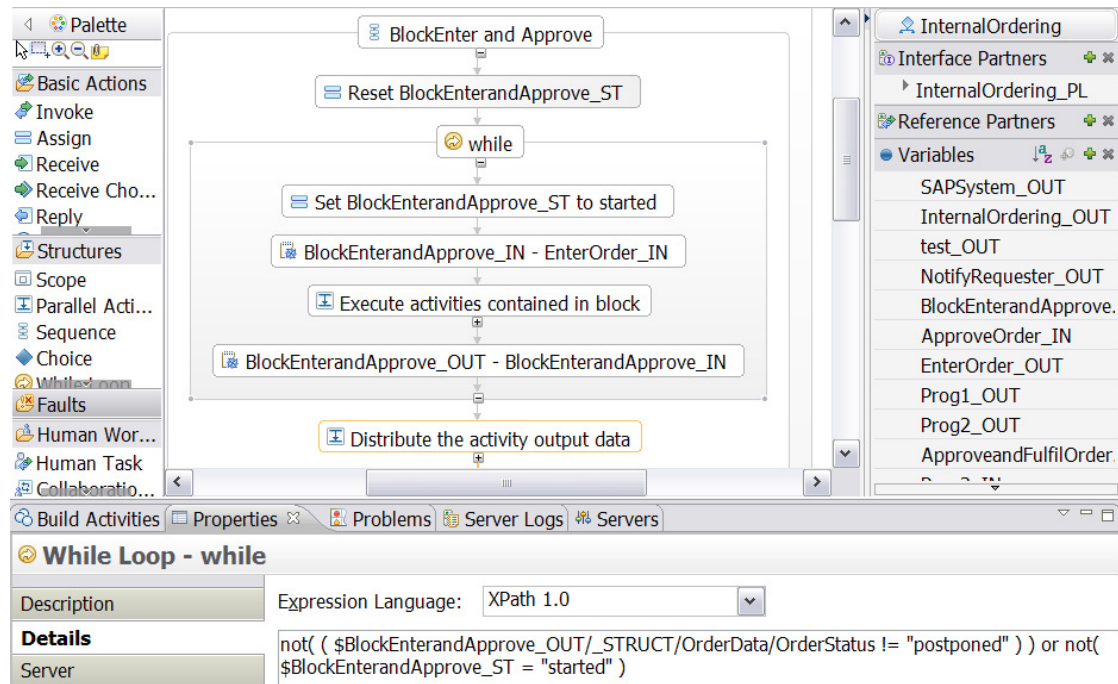


Figure 59: Mapping an exit condition expression to BPEL

Activity expiration

Activity expiration can be referred to in FDL conditions and is therefore a crucial task for migration from WebSphere MQ Workflow. Figure 60 shows a simple example that illustrates how exit and transition conditions can use the expiration event of activity "Get customer name".

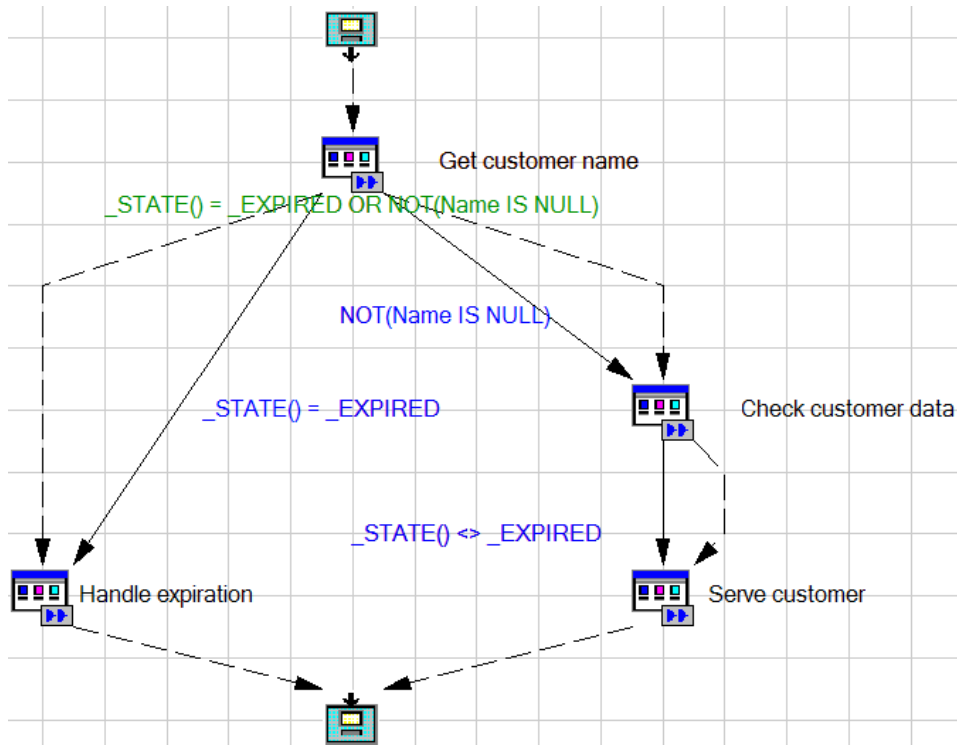


Figure 60: Activity expiration

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 FDL2BPEL Conversion uses "XPath" for the migrated condition expressions, you cannot directly refer to the corresponding "timeout" state in a BPEL process. That is why FDL2BPEL Conversion 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: $\$Act1_ST = "expired"$

Figure 61 to Figure 64 show the translation of activities with and without an exit condition:

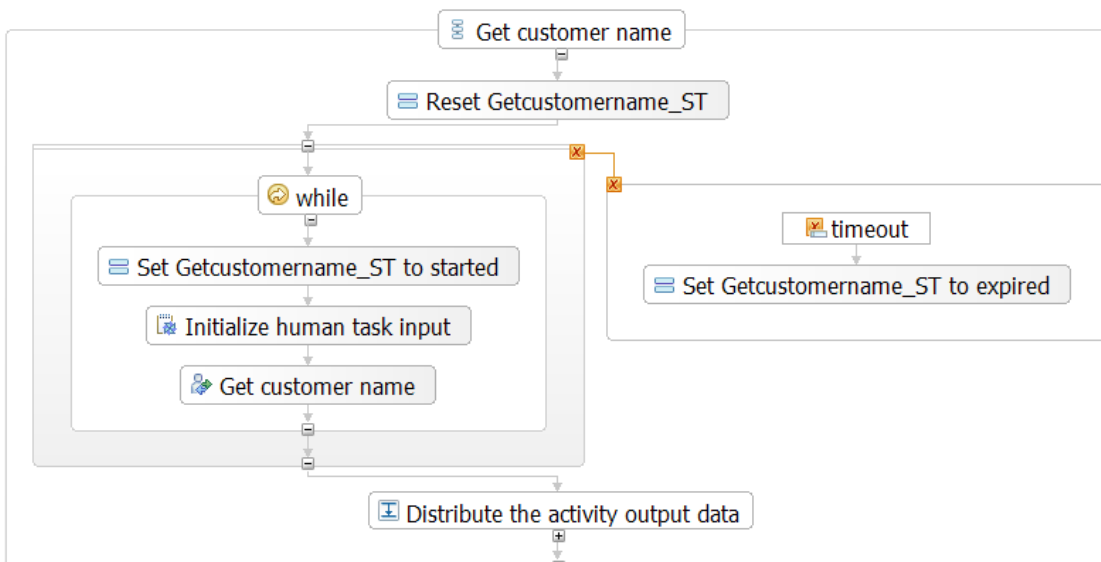


Figure 61: BPEL representation of FDL activity "Get customer name"

While Loop - while

Description	Expression Language: XPath 1.0
Details	not(((\$Getcustomername_ST = "expired") or not(not(\$Getcustomername_OUT/_STRUCT/Name)))) or not(\$Getcustomername_ST = "started")
Server	

Figure 62: Translated exit condition of FDL activity "Get customer name"

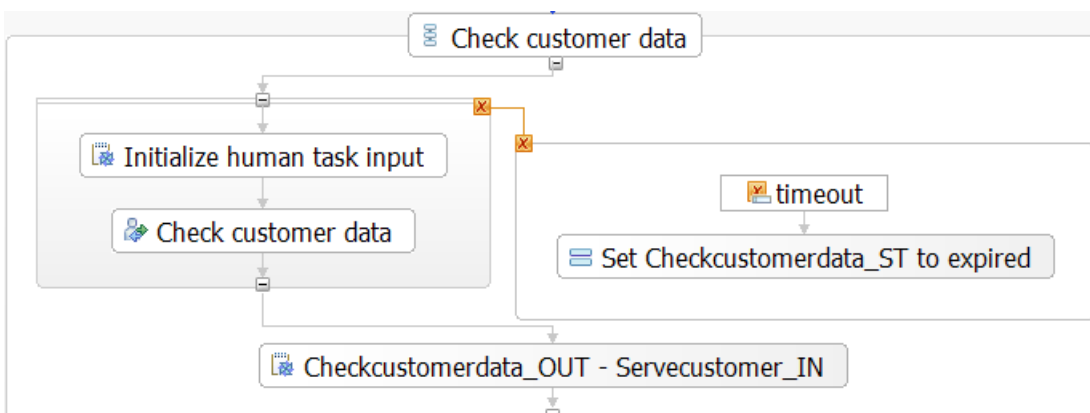


Figure 63: BPEL representation of FDL activity "Check customer data"

Link - Checkcustomerdata-to-Servecustomer

Description	Expression Language: XPath 1.0
Details	(\$Checkcustomerdata_ST != "expired")

Figure 64: Translated transition condition referring 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, FDL2BPEL Conversion 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":

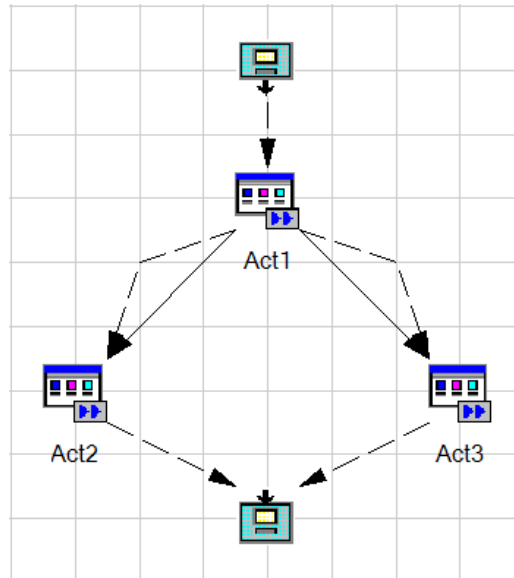


Figure 65: FDL data flow

Figure 66 shows part of the corresponding BPEL diagram view of the WID business process editor.

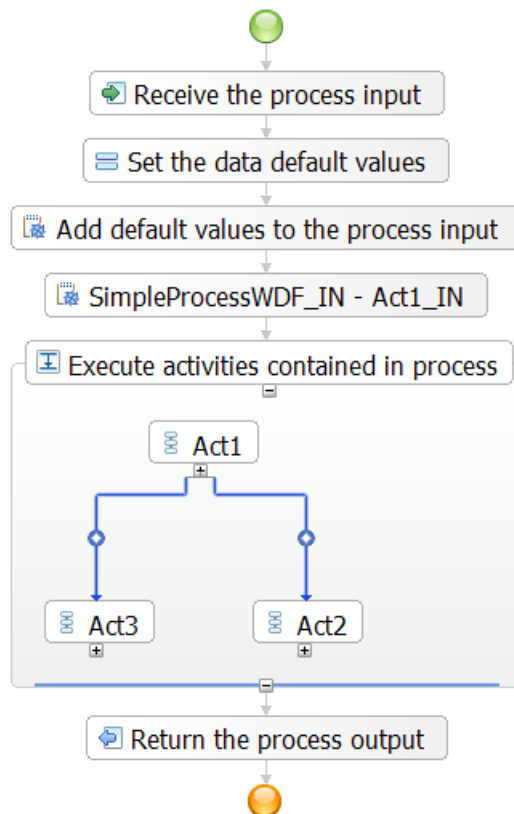


Figure 66: BPEL representation of FDL data flow (1)

The icons for activities "Act1", "Act2", and "Act3" indicate collapsed BPEL structured activities of type "Sequence". For each FDL activity, FDL2BPEL Conversion 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 addition, 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 57) and was therefore translated to a "human task":

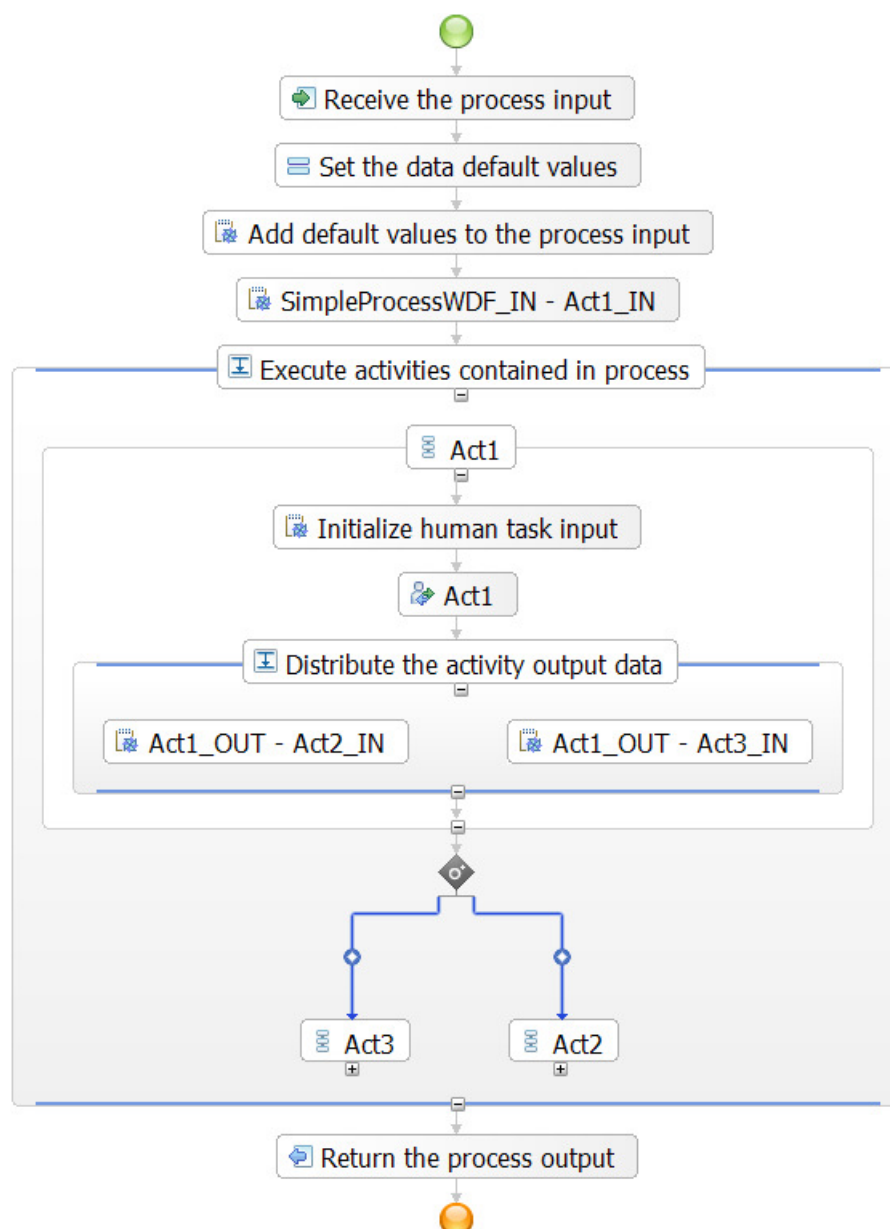


Figure 67: BPEL representation of FDL data flow (2)

Starting from top of the diagram, the "Assign" activity titled "Set the data default values" initializes any default values for the process output data and/or the data input and output of the activities contained in process "SimpleProcessWDF". The Java snippet "Add default values to the process input" completes the setting of initial values by the default values on the process input. An auxiliary variable ("SimpleProcessWDF_DEFAULT_IN") and the use of the

"WMQWFHelper.merge"-API⁵⁰ ensures that this activity does not overwrite the process input data. The 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 a Java snippet ("Initialize human task input") 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 ("Distribute the activity output data") 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"):

The screenshot shows the BPEL editor interface. The main canvas displays a process flow starting with an 'Invoke' activity, followed by 'Execute activities contained in process'. Inside this scope, there is an 'Act1' activity. Below 'Act1', there is a 'Distribute the activity output data' activity, which contains two parallel activities: 'Act1_OUT - Act2_IN' and 'Act1_OUT - Act3_IN'. The left sidebar shows the 'Build Activities' palette with various BPEL constructs. The right sidebar shows the 'Reference Partners' and 'Variables' list, including Act1_IN, Act1_OUT, Act2_IN, Act2_OUT, Act3_IN, Act3_OUT, SimpleProcessWDF_DEFAULT, SimpleProcessWDF_IN, and SimpleProcessWDF_OUT. The bottom pane shows the 'Snippet - Act1_OUT - Act2_IN' in the 'Java' view, containing the following code:

```
// Data flow from "Act1_OUT" to "Act2_IN"
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/Customer_DS/FirstName", Act2_IN,
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/Customer_DS/LastName", Act2_IN, "
if (Act2_IN == null) {
    com.ibm.websphere.bo.BOFactory boFactory = (com.ibm.websphere.bo.BOFactory) com.ibm.websphere.sc
    Act2_IN = boFactory.createByType(getVariableType("Act2_IN"));
}
```

Figure 68: Mapping FDL data mapping to a Java snippet

The snippet contains the following Java instructions:

⁵⁰ See Table 10: Semantic rules of "merge" operation on page 78.

```

Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(
    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"⁵¹ 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 10 summarizes the semantics of the "merge" operation:

⁵¹ Note that BPEL "Assign" activities behave like that. If data merging is not required, such as for setting default values, FDL2BPEL Conversion only uses "Assign" activities.

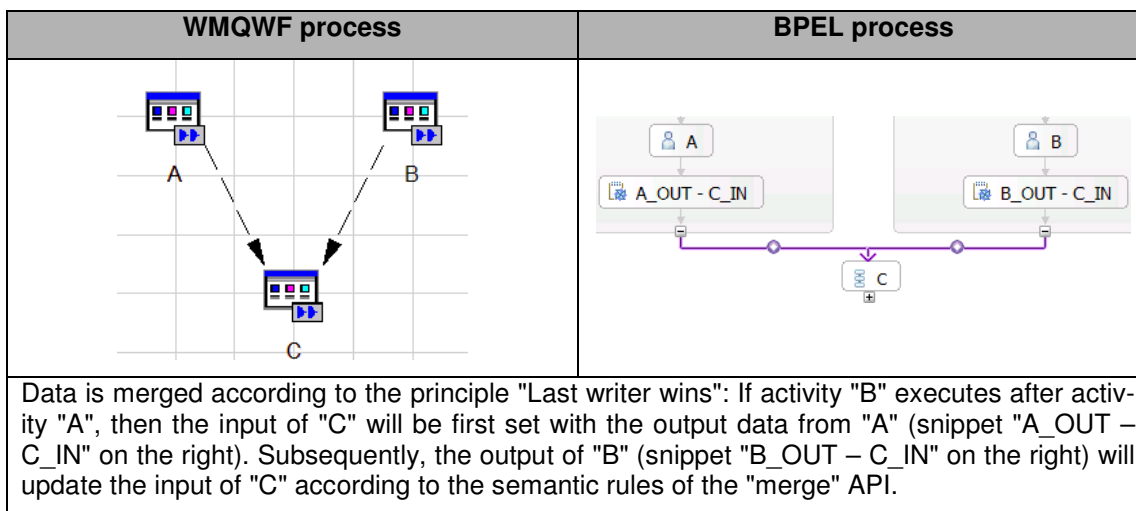
Table 10: Semantic rules of "merge" operation

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 11: Example 1 ("Merge" operation with two or more inbound data connectors)**Table 12: 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 13: Example 3 ("Merge" operation with inbound data connector and data loop connector)

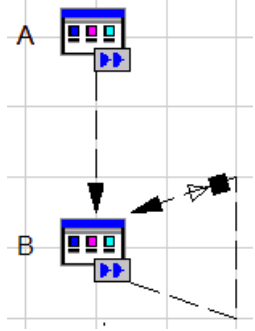
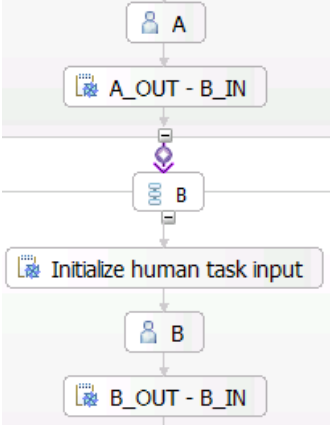
WMQWF process	BPEL process
	
<p>The input of activity "B" receives first data from the output of activity "A" (the snippet "A_OUT - B_IN"). Subsequently, activity "B" is executed and the output of "B" updates its own input data using the data loop connector (snippet "B_OUT - B_IN").</p>	

Table 14: 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 15: Example 5 (UPES or PEA⁵² activity ("staff activity") with default output data and/or default data connector)

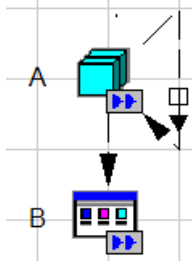
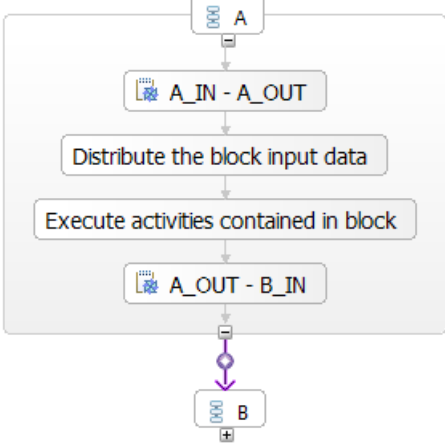
In this case, no data "merging" takes place. The activity result overwrites any preset data values in the output data container (in BPEL, the output variable).

Table 16: Example 6 (Process 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).

⁵² A PEA is a Program Execution Agent, which invokes a local program application. Migrating "PEA" activities is not supported (see migration hint "FDL program activity" on page 146). FDL2BPEL Conversion translates them to an "assumed staff activity" (see "Activity classification rules" on page 57).

Table 17: Example 7 (Block activity with default output data and/or default data connector)

WMQWF process	BPEL process
	
<p>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.</p>	

Mapping WMQWF staff assignment criteria to TEL people assignment criteria

Introduction

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

Table 18: 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.	<ul style="list-style-type: none"> • 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.
<p>Staff assignments can result from combining multiple staff assignment criteria. Examples:</p> <ul style="list-style-type: none"> • 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.
<p>WMQWF lets you specify staff assignment policies. Examples:</p> <ul style="list-style-type: none"> • 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

Staff repository

In WebSphere MQ Workflow, information about people is held in the Runtime database. The available attributes of people, organizations, and roles are built-in and cannot be extended by the user. WebSphere Process Server is more flexible and allows for different types of staff repositories. FDL2BPEL Conversion is limited to the migration of staff assignment criteria. For hints how to migrate the staff repository consult for instance the redbook "WebSphere MQ Workflow transition to WebSphere Process Server".⁵³

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. FDL2BPEL Conversion offers a translation of

⁵³ This redbook is available at <http://www.redbooks.ibm.com/redpieces/pdfs/sg247282.pdf>.

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 "default 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. FDL2BPEL Conversion maps all other WMQWF program activities to "empty activities" or "staff activities" according to the activity classification rules (see "Activity classification rules" on page 57). "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 FDL2BPEL Conversion, because its migration scope is limited to business processes, which should not depend on system migration aspects. FDL2BPEL Conversion 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 57)
3. staff assignment "inherited" is selected

Then FDL2BPEL Conversion 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 19 shows the mapping between WMQWF staff assignment criteria and TEL default people assignment criteria:

Table 19: 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	<tel:name>Group Members</tel:name> <tel:parameter id="GroupName"> %BPEL variable name\ _ACTIVITY_INFO/Organization% </tel:parameter> <tel:parameter id="IncludeSubGroups">true</tel:parameter>
People ⁵⁴	<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	<tel:name>Users by user ID without Named Users</tel:name>
Members of roles ⁵⁵	<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 organizations")	<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 notification may be given to the process administrator.

FDL2BPEL Conversion 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

⁵⁴ Restricted to the first three items from the people list field.

⁵⁵ Restricted to the first three items from the role list field.

classification rules (cp. section "Activity classification rules" on page 57). If you have specified a second notification for an activity of your WMQWF model, FDL2BPEL Conversion will translate it to an "escalation chain".

As an example, Figure 69 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.

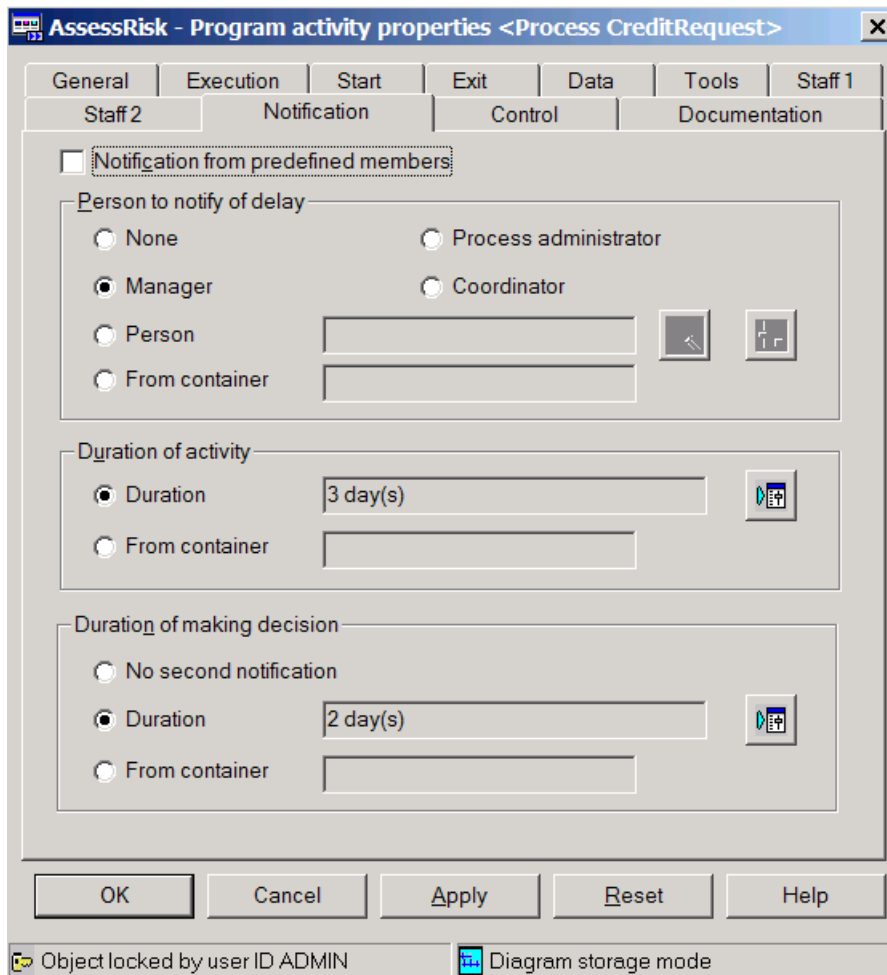


Figure 69: First and second notification of an FDL activity

The next two figures (Figure 70 and Figure 71) show how the first and second notifications are reflected in the WID human task editor as the members of an escalation chain:

AssessRisk_PTASK

▼ To-do Task

Name	AssessRisk_PTASK		
------	------------------	--	--

▼ Service Interface

▼ People Assignment (Receiver) + x

Administrators	Users by user ID	UserID *	%wf:process.starter%
Potential Owners	Users by user ID	UserID *	%wf:activity(CollectCreditInf

▼ User Interface + x

User Interface

▼ Escalation + x

Ready

Claimed Subtask started

- Escalation1
- Escalation2

Build Activities | Properties x | Problems | Server Logs | Servers

Escalation

Description	People assignment criteria: Manager of Employee by user ID
Details	Assigns the manager of an employee, given its user ID.

Assign People

Event Monitor

Global Event Settings

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

Figure 70: Mapping first notification to an escalation step of a human task

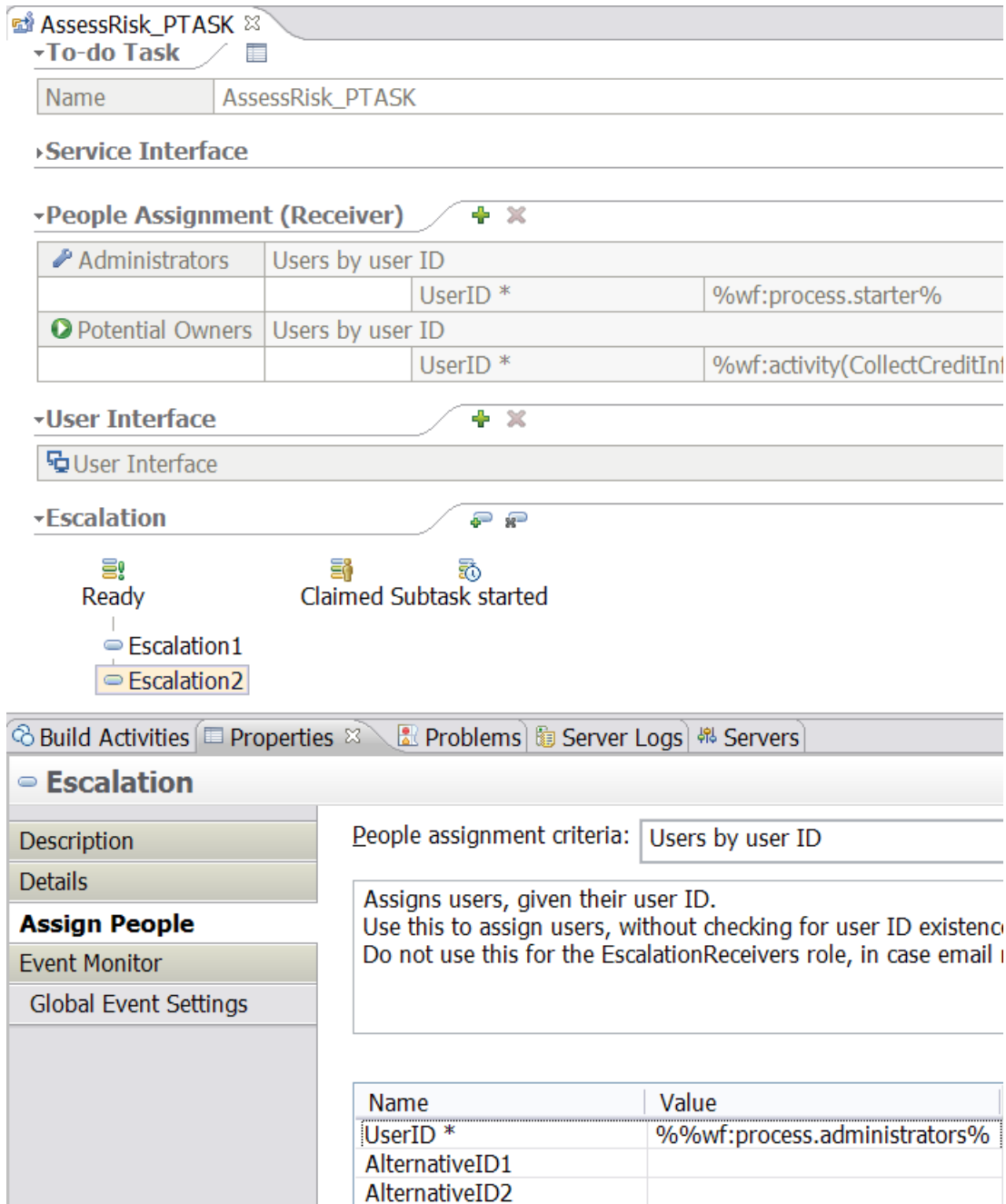


Figure 71: Mapping second notification to an escalation step of a human task

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.

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

Table 20: 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, FDL2BPEL Conversion 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:

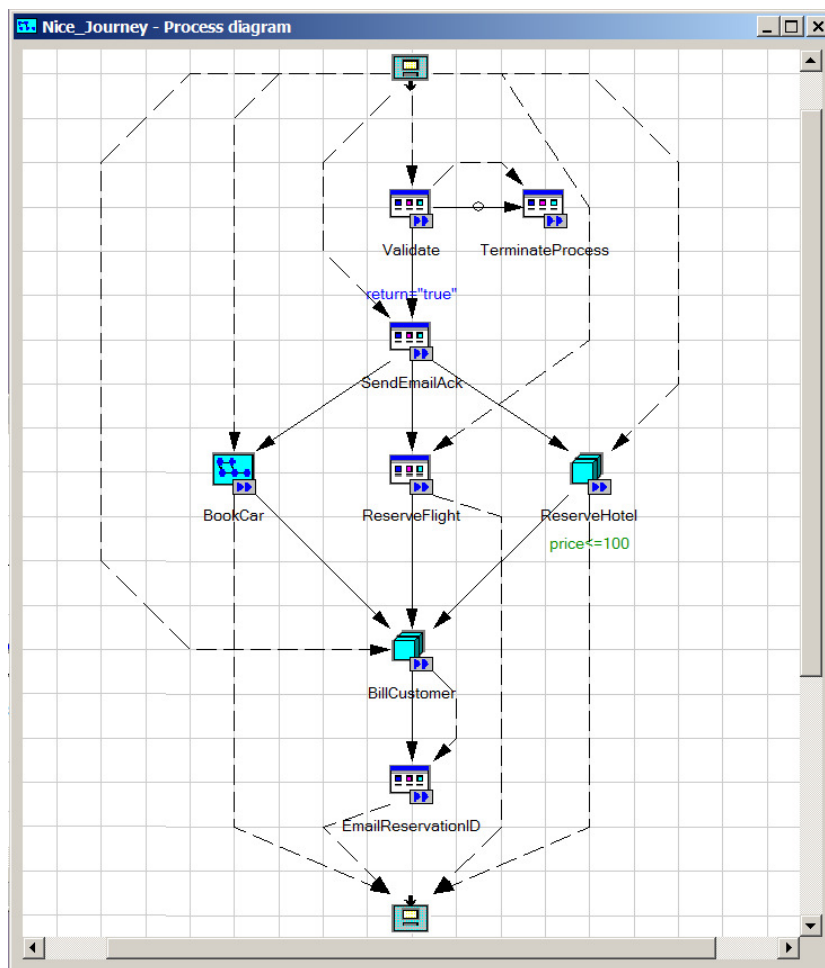


Figure 72: FDL process "Nice Journey"

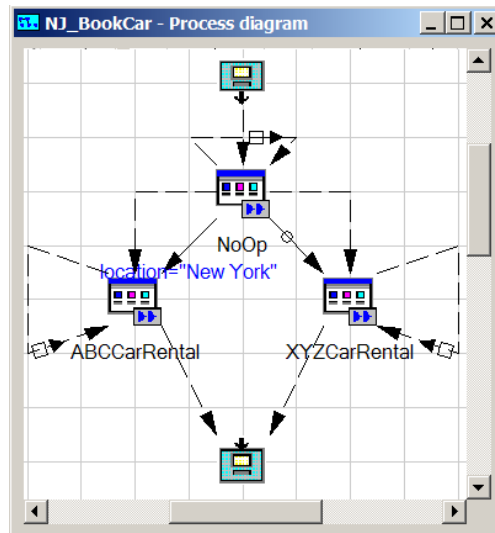


Figure 73: FDL subprocess "Book Car"

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:

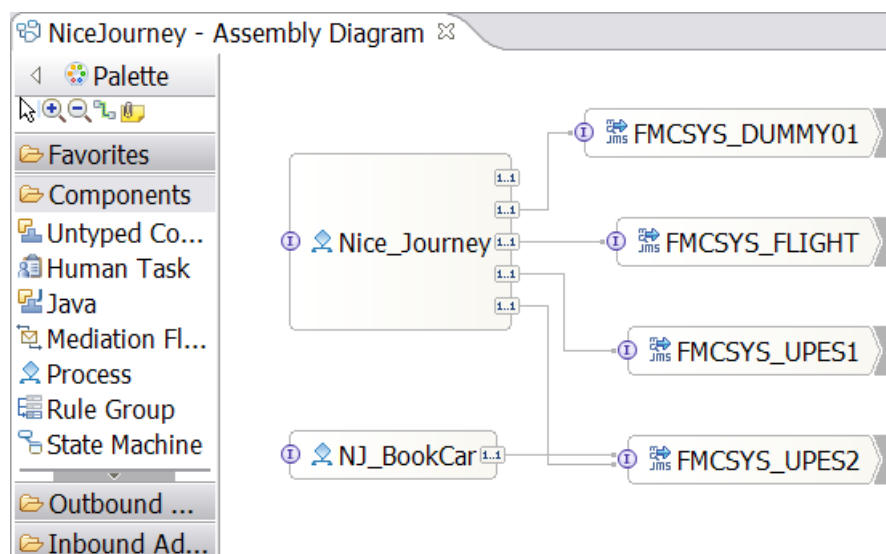


Figure 74: WID assembly diagram view

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, FDL2BPEL Conversion 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 "NiceJourney" and "NJ_BookCar".

As previously mentioned in "Mapping FDL PROCESS_ACTIVITY to BPEL" on page 64, FDL2BPEL Conversion 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 75) 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".

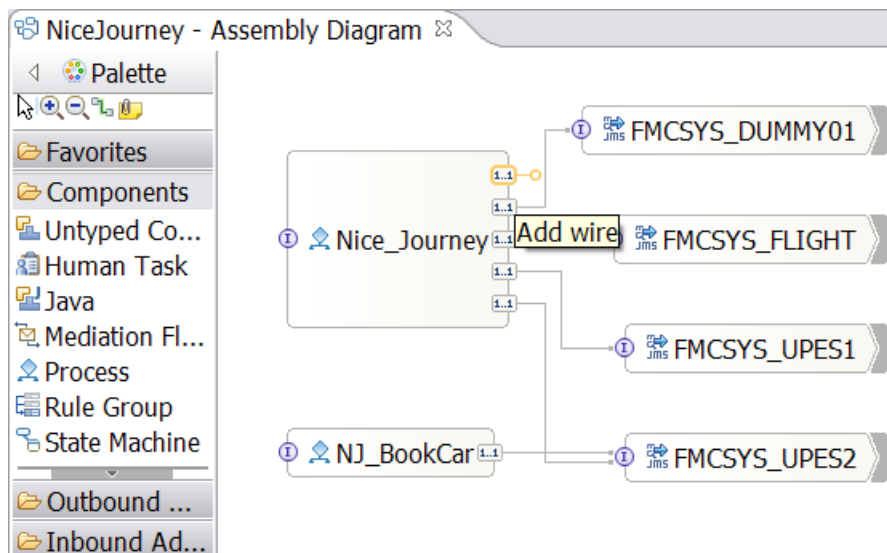


Figure 75: A deliberately omitted wire in the assembly diagram

SCA Imports

For each server definition in the FDL file, FDL2BPEL Conversion 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 40, 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*"⁵⁶ 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).

⁵⁶ See migration hint "XML message types" on page 143.

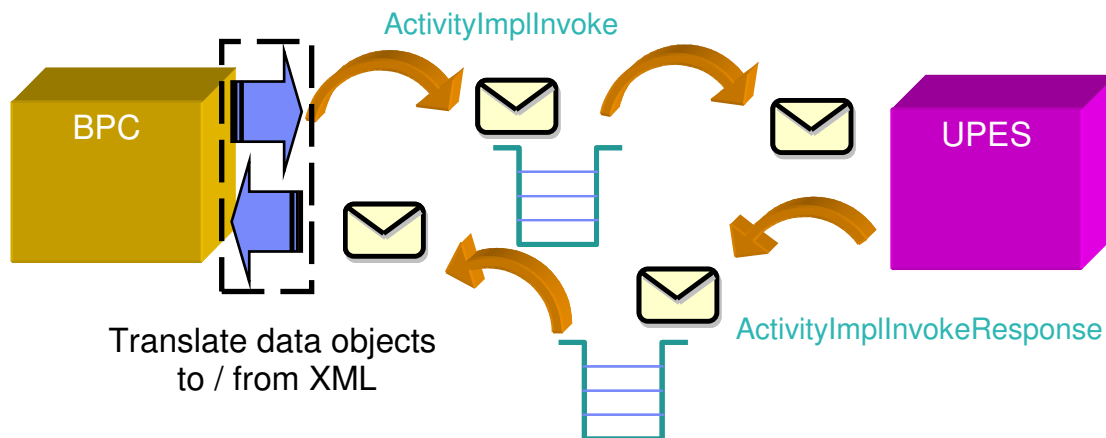


Figure 76: How BPC communicates with a UPES

FDL2BPEL Conversion 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 underlined 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/wsd/"
  xmlns:wsdl="http://www.ibm.com/xmlns/prod/websphere/scdl/wsd/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"
    </methodBinding>
  </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:

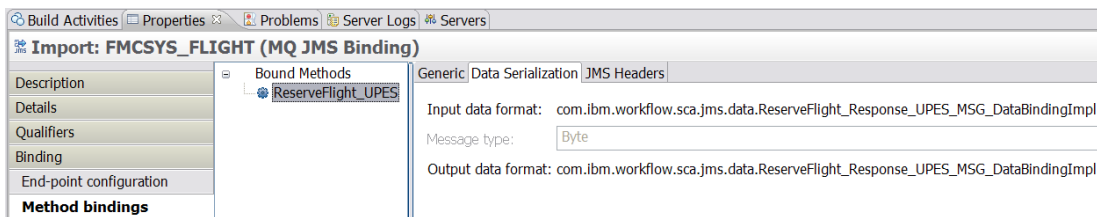


Figure 77: MQ JMS binding of an SCA import

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

```

D:\FDL2BPEL_TEST\NiceJourney_with_UPES\com\ibm\workflow\sca\jms\data\MakeReservationResponse_UPES_OUT_MSG_DataBindingImpl.java
----- 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.setDataTypes("MakeReservationResponse_UPES_OUT_MT");
        this.setDataTypeNamespace("http://www.ibm.com/xmlns/prod/websphere/mqwf/schema/");
    }
}

```

Figure 78: 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.

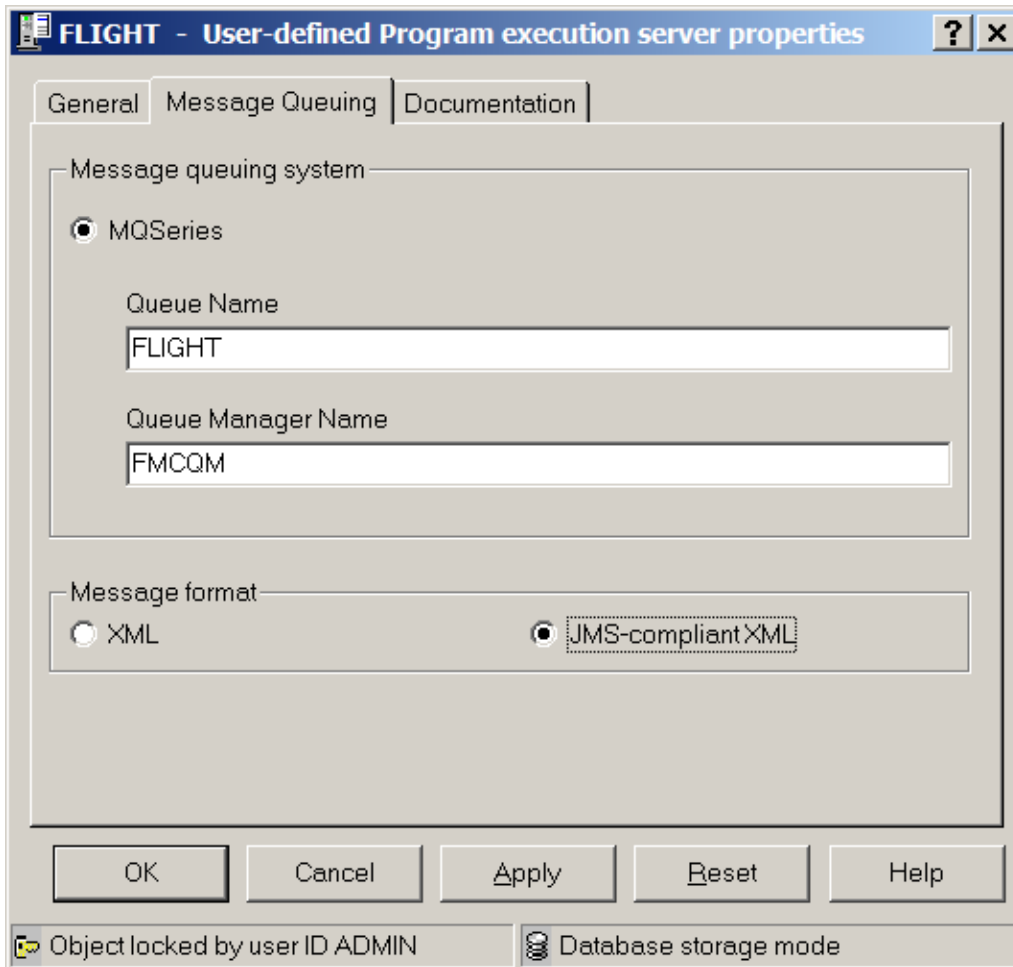


Figure 79: Queue manager in WMQWF Buildtime

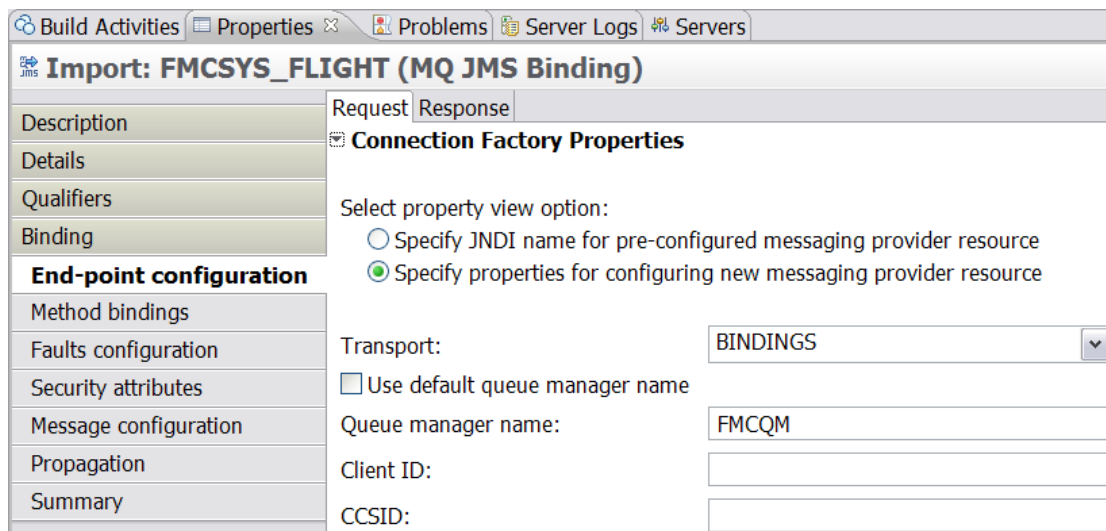


Figure 80: Queue manager in the WID assembly editor (properties view)

The screenshot shows the 'Import: FMCSYS_FLIGHT (MQ JMS Binding)' window in the WID assembly editor. The 'Properties' view is active, showing the 'Request' tab. The left sidebar contains a tree view with 'Binding' selected. The main area displays the 'Send Destination Properties' section, which is expanded. Under 'End-point configuration', the 'Specify properties for configuring new messaging provider resource' option is selected. The 'Type' is set to 'javax.jms.Queue'. The 'Base name' is 'JMS_FLIGHT'. The 'Target client type' is 'JMS'. The 'Digit Encoding Properties' section includes 'Use native encoding' (unchecked), 'Integer encoding' (Normal), 'Decimal encoding' (Normal), and 'Floating point encoding' (IEEENormal).

Figure 81: Request queue in the WID assembly editor (properties view)

The screenshot shows the 'Import: FMCSYS_FLIGHT (MQ JMS Binding)' window in the WID assembly editor. The 'Properties' view is active, showing the 'Response' tab. The left sidebar contains a tree view with 'Binding' selected. The main area displays the 'Receive Destination Properties' section, which is expanded. Under 'End-point configuration', the 'Specify properties for configuring new messaging provider resource' option is selected. The 'Type' is set to 'javax.jms.Queue'. The 'Base name' is 'JMS_FLIGHT_REPLY'. The 'Target client type' is 'JMS'. The 'Digit Encoding Properties' section includes 'Use native encoding' (unchecked), 'Integer encoding' (Normal), 'Decimal encoding' (Normal), and 'Floating point encoding' (IEEENormal).

Figure 82: Response queue in the WID assembly editor (properties view)

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

Table 21: 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 22: 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 83 below).

The screenshot displays the IBM WebSphere MQ Workflow Designer interface. The top part shows an Assembly Diagram for 'NiceJourney' with several components: 'Nice_Journey', 'NJ_BookCar', 'FMCSYS_DUMMY01', 'FMCSYS_FLIGHT', 'FMCSYS_UPES1', and 'FMCSYS_UPES2'. The bottom part shows the 'Properties' pane for the 'Import: FMCSYS_FLIGHT (MQ JMS Binding)'. The 'Interfaces' tree view is expanded to show 'FMCSYS_FLIGHT_PT'. The 'Details' tab is selected, showing the 'Preferred interaction style' property set to 'Asynchronous'.

Figure 83: 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 provided that only the following WMQWF message types are used:

- 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 SP7. 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)⁵⁷.

⁵⁷ 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.

FDL2BPEL Conversion 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 FDL2BPEL Conversion generates two BPEL variables and a snippet activity that does the data mapping.

You might want to run FDL2BPEL Conversion with optimization options in order to improve its performance by either running FDL2BPEL with the following optimization options:

- Merge Java snippets (command line option **-mj**)
- Remove unnecessary structural elements (command line option **-es**)
- Allow sharing variables (command line option **-ev**)

Selecting option "Optimize the business model during migration" (command line option **-opt**) is equivalent to selecting all three optimization options at once. Note, that FDL2BPEL always applies option "Merge Java snippets" before option "Remove unnecessary structural elements".

Merging Java snippets

For details how to use this option see chapter "Using as WID Migration Wizard" on page 18 for the WID Migration wizard and chapter "Using optimization options" on page 26 if you run FDL2BPEL Conversion from the command line.

Adjacent Java snippets that you would like to merge into one snippet may occur in the context of the following two different scenarios:

- Java snippets as parallel activities (contained in a "Parallel Activities" node)
- Java snippets as sequential activities (contained in a "Sequence" activity)

Accordingly, FDL2BPEL offers two optimization options:

- Merge parallel Java snippets (command line option **-mjp**)
- Merge sequential Java snippets (command line option **-mjs**)

Selecting option "Merge Java snippets" (command line option **-mj**) is equivalent to selecting both options together.

Merging parallel Java snippets

You may encounter parallel Java snippets in "Parallel Activities" with the following captions:

- "Distribute the input data of process ... to the start activities of its implementation"
- "Distribute the input data of block activity ... to the start activities of its implementation"
- "Distribute the activity output data"

For example, assume that you have an FDL activity "Act1" with two outbound data connectors as already shown in Figure 65 on page 74. As explained in chapter "Mapping FDL data flow to

BPEL" on page 73 the outbound data connectors are mapped to Java snippets "Act1_OUT - Act2_IN" and "Act1_OUT – Act3_IN" that are contained in a "Parallel Activities" node with caption "Distribute the activity output data" (see Figure 68 on page 76). Running FDL2BPEL Conversion with option "Merge parallel Java snippets" (command line option **-mjps**) eliminates the "Distribute the activity output data" node and merges the two Java snippets into a single one (see Figure 84).

The screenshot displays the IBM Business Process Manager interface. The top pane shows a BPEL diagram with a sequence of activities: 'Act1', 'Initialize human task input', 'Act1', and 'Distribute the activity output data'. This sequence is followed by a choice activity that branches into 'Act3' and 'Act2'. The bottom pane shows the 'Snippet - Distribute the activity output data' in the 'Java' view. The snippet contains the following code:

```

// Distribute the activity output data
// Data flow from "Act1" to "Act2"
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/
Act2_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/
if (Act2_IN == null) {
    com.ibm.websphere.bo.BOFactory boFactory = (com.ibm.websphere.b
    Act2_IN = boFactory.createByType(getVariableType("Act2_IN"));
}

// Data flow from "Act1" to "Act3"
Act3_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT"
Act3_IN = com.ibm.bpe.interop.WMQWFHelper.merge(Act1_OUT, "_STRUCT/
if (Act3_IN == null) {
    com.ibm.websphere.bo.BOFactory boFactory = (com.ibm.websphere.b
    Act3_IN = boFactory.createByType(getVariableType("Act3_IN"));
}

```

Figure 84: Distributing the activity output data with a merged Java snippet

Merging sequential Java snippets

Sequentially arranged Java snippets occur as members of BPEL "Sequence" activities as shown on the left-hand side of Figure 85. Note that the "While" activity has a single "Sequence" child node which is not displayed by WID, such that you only see the contained members of the invisible "Sequence" activity. On the right-hand side you see the same "While" activity after running FDL2BPEL with option "Merge sequential Java snippets" (command line option **-mjs**). As you will notice, the adjacent Java snippets that precede the "Invoke" activity "BookHotel" are combined into the new snippet "Setup activity". The Java snippets that follow the activity "BookHotel" are replaced by the snippet "Shutdown activity".

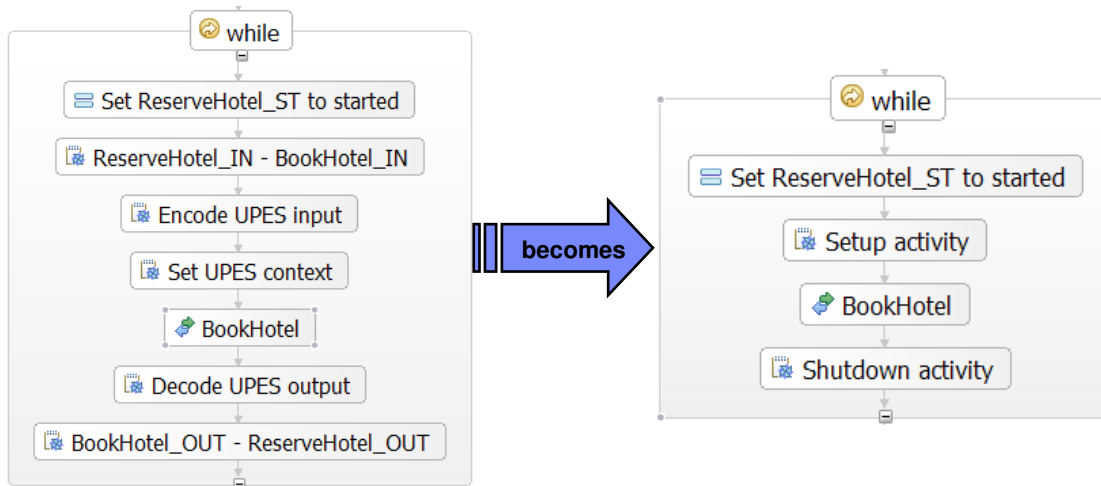


Figure 85: Merging sequentially arranged Java snippets

Removing unnecessary structural elements

Examples of unnecessary structural elements are:

- “Sequence” or “Parallel activities” node that contains a single child node
- Nested “Sequence” activities
- “Parallel activities” node that contains a simply linked activity chain

The first two cases should never occur in BPEL process models migrated from WMQWF, because FDL2BPEL Conversion automatically takes care of such scenarios. But the case of a simply linked activity chain is not detected automatically without enabling the option “Remove unnecessary structural elements” (command line option **-es**). Figure 86 shows an example of a simply linked activity chain (without any data connector) in WMQWF Buildtime. The diagram on the right represents the inner process structure of the activity “Block” on the left.

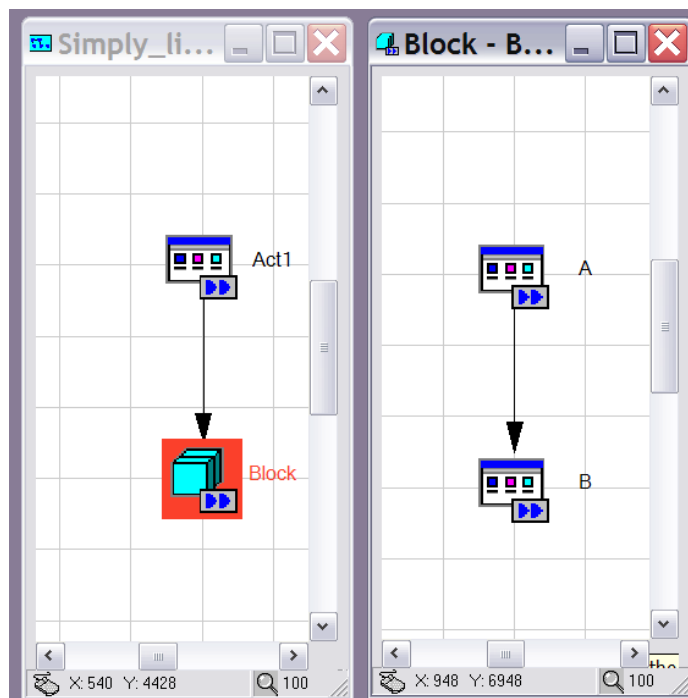


Figure 86: Simply linked activity chain (WMQWF Buildtime)

Figure 87 shows how FDL2BPEL maps the FDL process to BPEL.

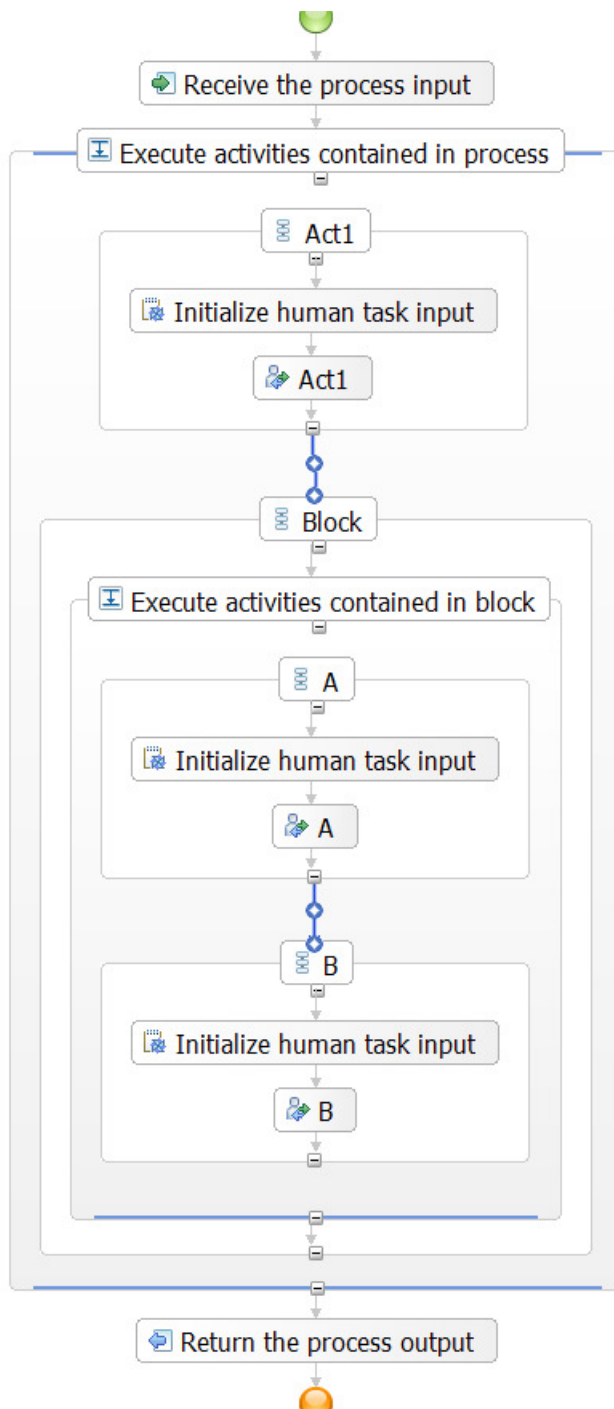


Figure 87: Simply linked activity chain (WID process editor)

It is quite evident that the generated “Parallel Activities” nodes and “Sequence” activities are not necessary and that the contained activities are better combined into a single “Sequence” activity. The result of running FDL2BPEL with option “Remove unnecessary structural elements” (command line option **-es**) is shown in Figure 88.⁵⁸

⁵⁸ Note that the activities are now contained in one “Sequence” activity which is hidden.

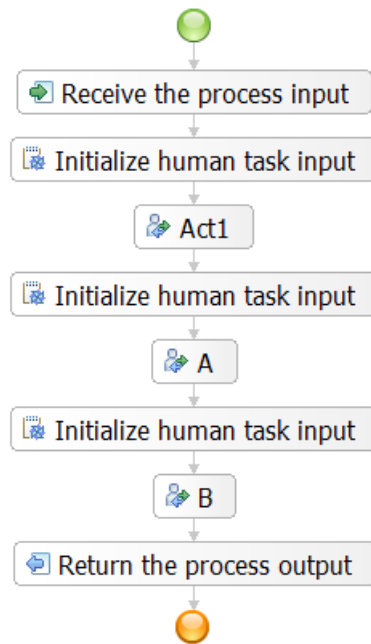


Figure 88: BPEL process after using option “Remove unnecessary structural elements”

Sharing variables

Overview

FDL2BPEL Conversion lets you reduce the number of generated BPEL variables by “variable sharing”. “Variable sharing” means that the input variable of one activity also serves as output variable of another activity, provided that the participating activities exchange data of the same data type by a „container to container“-mapping (see Figure 89).

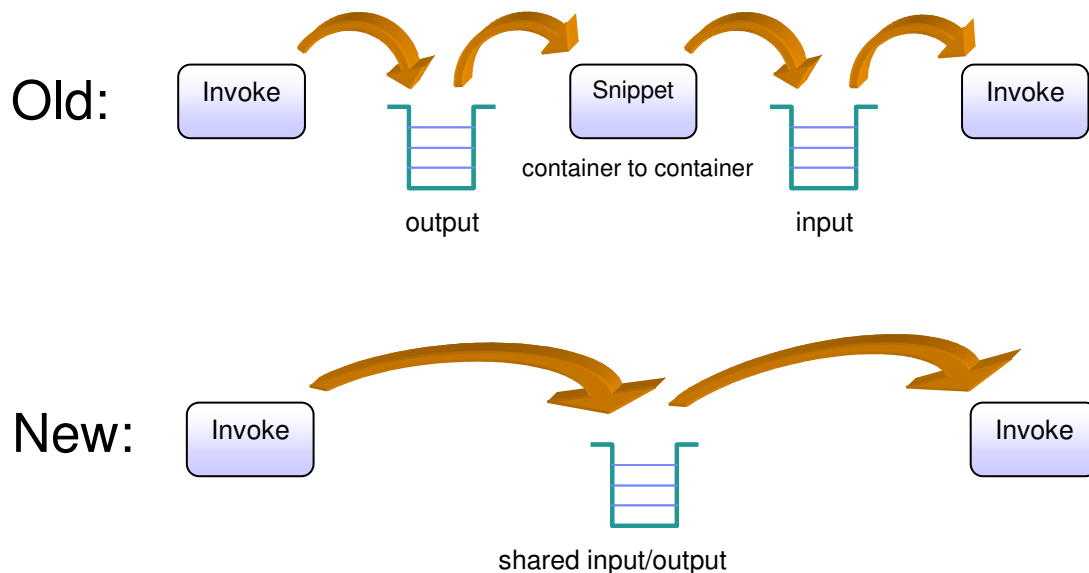


Figure 89: Sharing of BPEL variables

That means the output message is passed as a whole information entity without modifications. Note that a „container to container“-mapping means in terms of WMQWF the following set of FDL statements⁵⁹:

```
DATA
  FROM name of data connector source TO name of data connector target
  MAP '_PROCESS_INFO' TO '_PROCESS_INFO'
  MAP '_ACTIVITY_INFO' TO '_ACTIVITY_INFO'
  MAP '_STRUCT' TO '_STRUCT'
```

An important postulate is that variable sharing must not modify the data flow as it was specified in the original FDL process model. In other words, sharing variables must neither change the order of data flow execution nor change the data flow routing.

Sharing BPEL variables requires a careful treatment, because there are subtle dependencies between data flow and control flow that must not be disarranged. Otherwise the sharing of BPEL variables might change the results of affected business process operations thereby invalidating the model migration from WMQWF to WPS.

For instance, joining the input of activities by a shared BPEL variable might be dangerous, if the target activities belong to parallel control flow paths. Data default or loop connectors added to these activities would propagate the variable sharing on parallel threads such that originally independent operations can mutually overwrite their data input or output in an unpredictable manner.

FDL2BPEL applies a heuristic approach that ensures data flow integrity except for the use of program applications that access variables by API. You should note that some opportunities for optimization might not be exploited. In particular, FDL2BPEL does not address improvements that might be achieved by reassigning data structures to BPEL variables such that they no longer correspond to the original FDL data containers. Testing with typical customer process models have shown that you can expect a proportion of 2% dispensable variables on average. If you use option **-pn** ("Do not create predefined data members") you can expect about 20% dispensables variables on average (3 - 40% depending on the model characteristics).

Example

The following example consists of two simple activity chains contained in a WMQWF process (see Figure 90). It is assumed that all data connectors have a "container to container" mapping as described above.

⁵⁹ By default you only have a '_STRUCT' TO '_STRUCT' mapping when drawing a data connector.

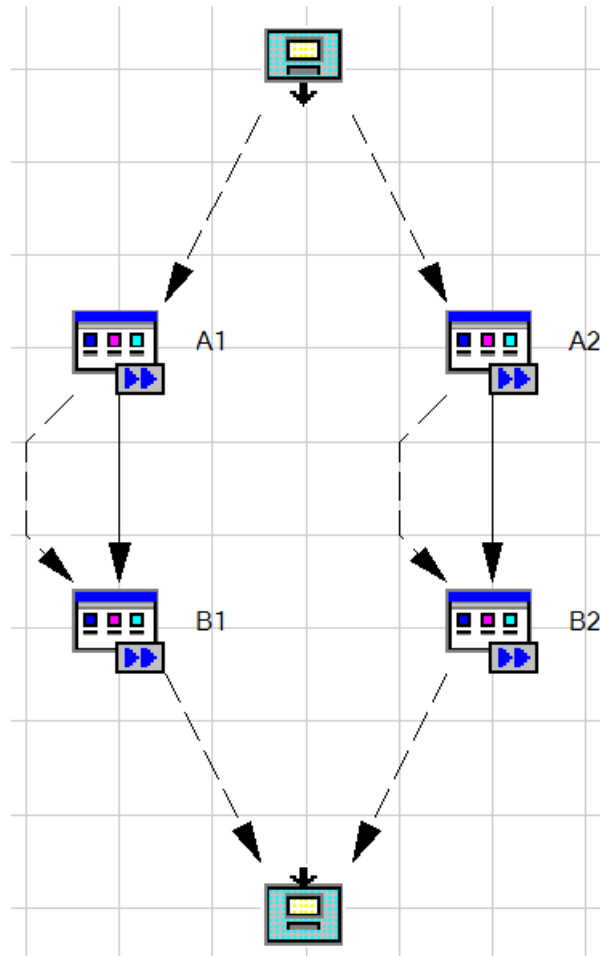


Figure 90: WMQWF process with two simple activity chains

As described in chapter “Mapping FDL data container to BPEL” on page 51, FDL2BPEL maps the input/output containers to BPEL variables. Accordingly, the tool generates in total eleven BPEL variables as you can verify in Figure 91.

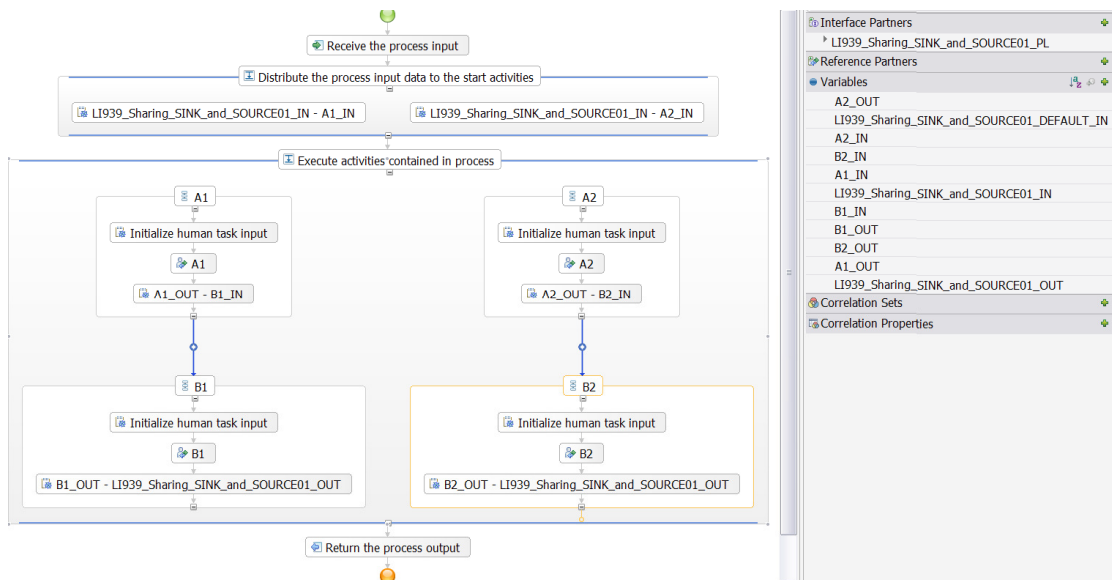


Figure 91: Migrated activity chains without optimization

Migrating the same WMQWF process model with option “Allow sharing variables” (command line option **-ev**) produces a BPEL process model with only five BPEL variables (see Figure 92).

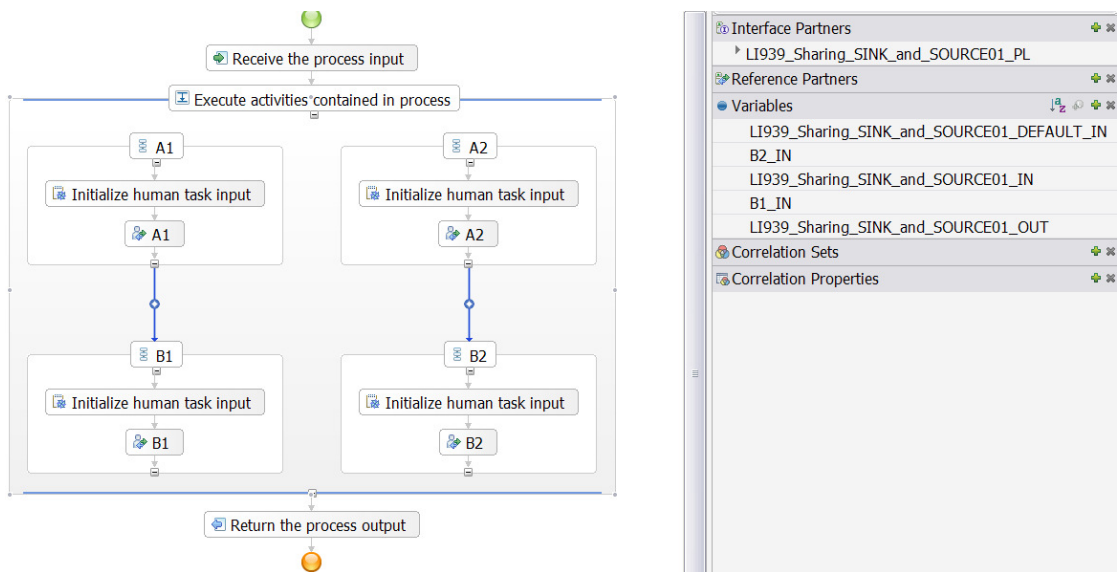


Figure 92: Migrated activity chains with shared BPEL variables

Note that the improved BPEL model also needs fewer Java snippets.

Using additional optimization settings (for experts only)

The optimization algorithm for sharing variables observes the following preconditions (among others):

- common data types
- no default data values on the inputs and outputs of activities
- no dynamically assigned activity properties (taken from predefined data members in the input)

Sometimes, default data values and dynamically assigned properties are specified in the model without them ever being used. Therefore the opportunities for sharing variables can be increased if the optimization algorithm is made to tolerate such properties.

Analyze the activity settings of your model with care and let the optimization algorithm tolerate the following model properties if you are sure that this will not have unwanted side effects on the runtime behavior of the business model:

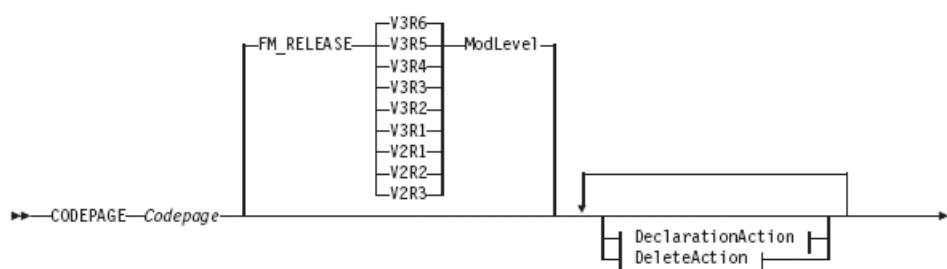
Option	Hint
Tolerate the default data values on the inputs and outputs of activities (command line option -id)	Use this option if you are sure that initial data values in your business process model do not affect the control flow of your workflow.
Tolerate the dynamically assigned activity properties (command line option -ip)	Use this option if you are sure that your WMQWF model does not exploit settings taken from predefined data members. Consider deselecting option Create predefined data members (command line option -pn) instead of -ip , which further reduces unneeded overhead in the migrated model (see chapter “Mapping an FDL data container to an XML schema definition” on page 37).
Tolerate dynamically assigned staff (command line option -ips)	Use this option if you are sure that your WMQWF model does not exploit staff assignment criteria taken from predefined data members.
Tolerate dynamically assigned notification (command line option -ipn)	Use this option if you are sure that your WMQWF model does not exploit notification settings taken from predefined data members.
Tolerate dynamically assigned priority (command line option -ipp)	Use this option if you are sure that your WMQWF model does not exploit priority settings taken from predefined data members.

Appendix 1: Migration coverage listed by FDL syntax expressions

Introduction

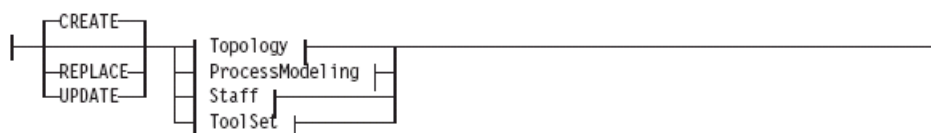
Not all constructs that you can specify in FDL can be mapped in a one-to-one fashion to a corresponding BPEL construct due to the differences in the programming models. Appropriate warnings in the output of FDL2BPEL Conversion are provided as XML comments in the generated artifacts (such as the files with the extension ".bpe1"). These warnings inform you about constructs that are not migrated in a one-to-one fashion. This chapter and the following⁶⁰ 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 workaround possible	See page
CODEPAGE Codepage		X ⁶¹			
FM_RELEASE VnRn ModeLevel		X ⁶²			
DeclarationAction					105
DeleteAction			X ⁶³		

DeclarationAction



⁶⁰ See "Appendix 2: Migration hints" on page 131.

⁶¹ See "FDL with codepage different from system codepage" on page 131.

⁶² See "Migration of early FDL versions" on page 131.

⁶³ See "FDL processing actions" on page 131.

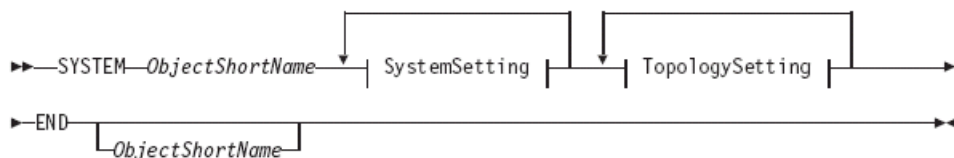
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
CREATE			X ⁶⁴		
REPLACE			X ⁶⁴		
UPDATE			X ⁶⁴		
<u>Topology</u>					106
<u>ProcessModeling</u>					112
Staff			X ⁶⁵		
ToolSet				X ⁶⁶	

Topology



FDL syntax expression ⁶⁷	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		
<u>System</u>					106
<u>Server</u>					110
ProgramExecutionAgent				X	
QueueManager			X		

System



⁶⁴ See "FDL processing actions" on page 131.

⁶⁵ See "WMQWF staff repository" on page 138.

⁶⁶ See "Migration of WMQWF Buildtime tool set definitions not supported" on page 132.

⁶⁷ See "System network" on page 144.

FDL syntax expression ⁶⁸	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		
<i>TopologySetting</i>					107

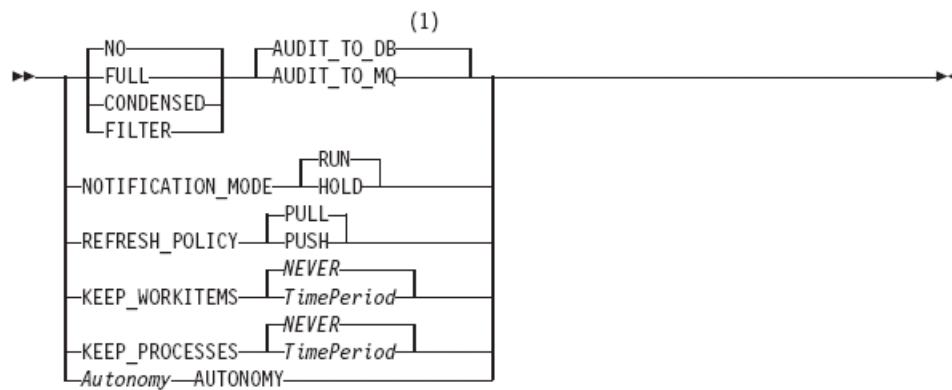
TopologySetting



⁶⁸ See "System network" on page 144.

FDL syntax expression ⁶⁹	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 <i>OperationSetting</i>			x		
SESSION <i>SessionSetting</i>			x		
SERVER <i>DefaultServerSetting</i>			x		
PROGRAM_EXECUTION_AGENT <i>DefaultProgramExecutionAgentSetting</i>			x		
PROCESS <i>DefaultProcessSetting</i>					108
ACTIVITY <i>DefaultActivitySetting</i>					110
PROGRAM <i>DefaultProgramSetting</i>			x		
IMPORT <i>DefaultImportSetting</i>			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*.

⁶⁹ See "Properties inherited from domain or system" on page 144.

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 ⁷⁰			
NOTIFICATION_MODE RUN	x				
NOTIFICATION_MODE HOLD				x ⁷¹	
REFRESH_POLICY PULL	x				
REFRESH_POLICY PUSH				x ⁷²	
KEEP_WORKITEMS (NEVER <u>TimePeriod</u>)				x ⁷³	
KEEP_PROCESSES (NEVER <u>TimePeriod</u>)	x ⁷⁴				
<u>TimePeriod</u>					130
<u>Autonomy</u> AUTONOMY					109

Autonomy



FDL syntax expression ⁷⁵	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				

⁷⁰ See "Audit settings" on page 145.

⁷¹ See "Notification mode" on page 138.

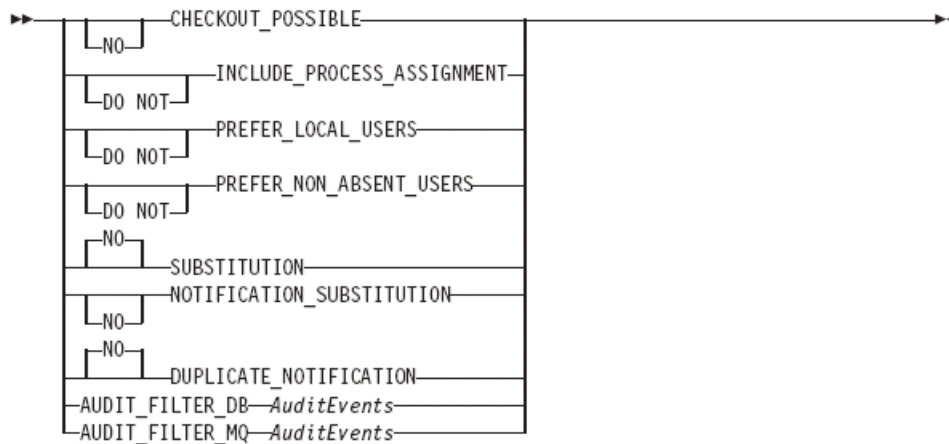
⁷² See "Workitem refresh policy" on page 147.

⁷³ See "Policy for how to deal with finished workitems" on page 147.

⁷⁴ See "Policy for how to deal with finished process instances" on page 148.

⁷⁵ See "Process autonomy modes" on page 145.

DefaultActivitySetting



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
(NO) CHECKOUT_POSSIBLE	x ⁷⁶				
INCLUDE_PROCESS_ASSIGNMENT				x ⁷⁷	
DO NOT INCLUDE_PROCESS_ASSIGNMENT	x				
PREFER_LOCAL_USERS				x ⁷⁷	
DO NOT PREFER_LOCAL_USERS	x				
(DO NOT) PREFER_NON_ABSENT_USERS	x				
(NO) SUBSTITUTION	x				
NOTIFICATION_SUBSTITUTION				x ⁷⁸	
NO NOTIFICATION_SUBSTITUTION	x				
DUPLICATE_NOTIFICATION				x ⁷⁸	
NO DUPLICATE_NOTIFICATION	x				
AUDIT_FILTER_DB <i>AuditEvents</i>		x ⁷⁹			
AUDIT_FILTER_MQ <i>AuditEvents</i>		x ⁷⁹			

Server



⁷⁶ Serves as indicator for a "staff activity" (See "Activity classification rules" on page 57).

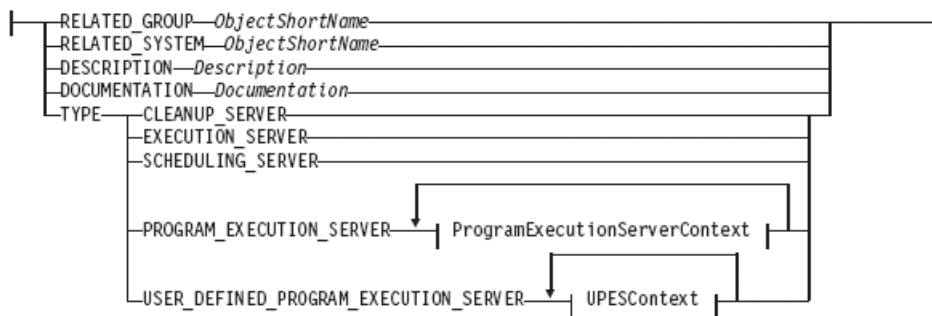
⁷⁷ See "Staff assignment criteria" on page 141.

⁷⁸ See "Notification policy" on page 138.

⁷⁹ See "Audit settings" on page 145.

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 <i>ObjectShortName</i>	X ⁸⁰				
<u>ServerSetting</u>					111

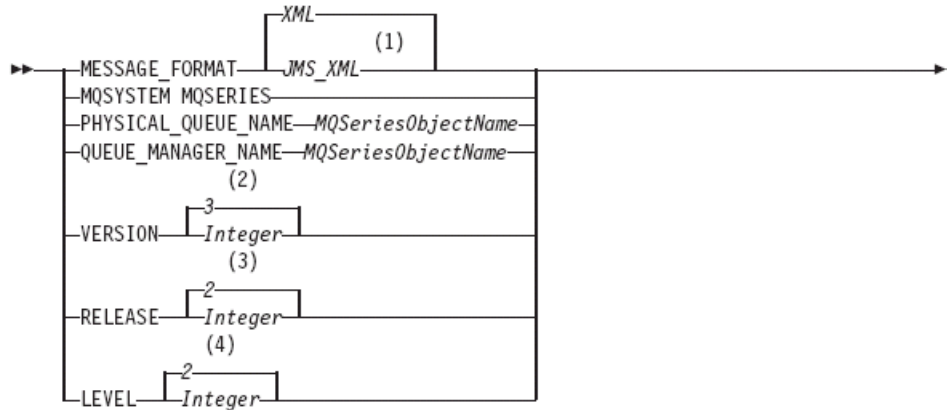
ServerSetting



FDL syntax expression ⁸⁰	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 <i>ObjectShortName</i>	x				
RELATED_SYSTEM <i>ObjectShortName</i>	x				
DESCRIPTION <i>Description</i>				x	
DOCUMENTATION <i>Documentation</i>				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 <u>UPESContext</u>					112

⁸⁰ See "User-defined program execution server (UPES)" on page 143.

UPESContext

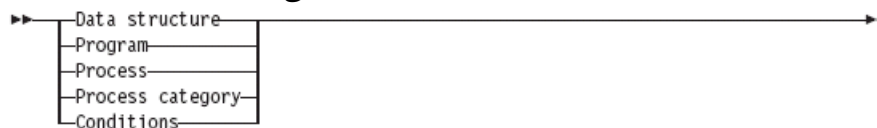


Notes:

- 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.

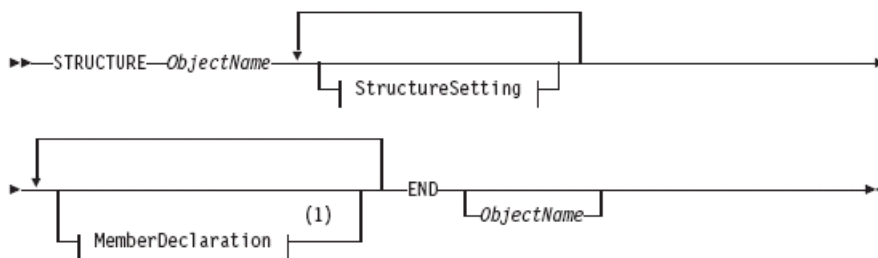
FDL syntax expression	Migration supported with FDL2BPPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
MESSAGE_FORMAT XML	x				
MESSAGE_FORMAT JMS_XML	x				
MQSYSTEM MQSERIES				x	
PHYSICAL_QUEUE_NAME MQSeriesObjectName	x				
QUEUE_MANAGER_NAME MQSeriesObjectName	x				
VERSION (3 Integer)				x	
RELEASE (2 Integer)				x	
LEVEL (2 Integer)				x	

ProcessModeling



FDL syntax expression ^{81 82}	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
<u>Data structure</u>					113
<u>Program</u>					114
<u>Process</u>					115
<u>Process category</u>					128
<u>Conditions</u>					129

Data structure



StructureSetting:



MemberDeclaration:

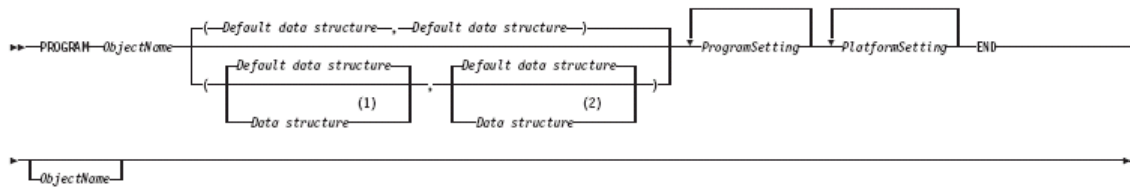


⁸¹ See "User-defined program execution server (UPES)" on page 143.

⁸² See "XML message types" on page 143.

FDL syntax expression ⁸³	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 ⁸³			
MemberDeclaration				X ⁸⁴	

Program



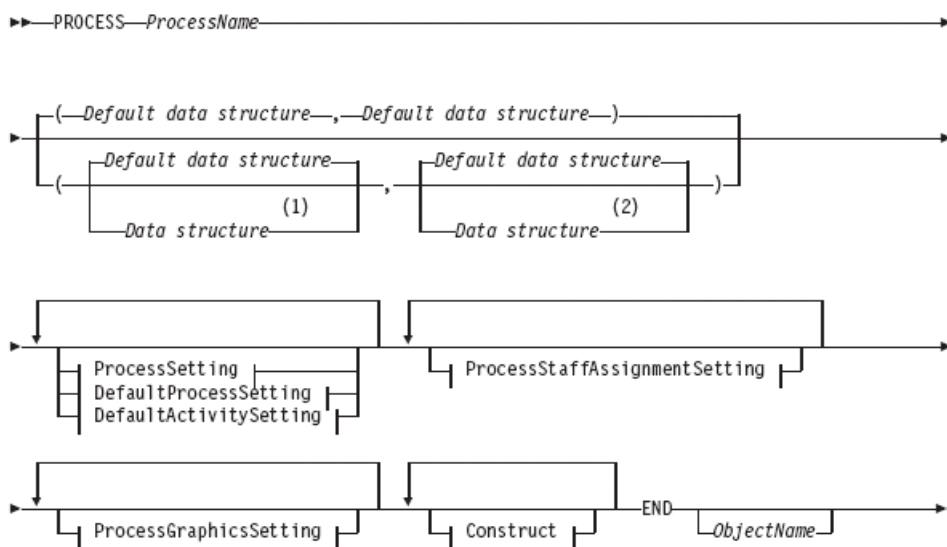
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROGRAM <i>ObjectName</i>	X				
<i>Default data structure</i>	X				
<i>Data structure</i>					113
ProgramSetting				X ⁸⁵	
PlatformSetting				X ⁸⁵	

⁸³ See "Data description and documentation" on page 144.

⁸⁴ See "Data member description and documentation" on page 144.

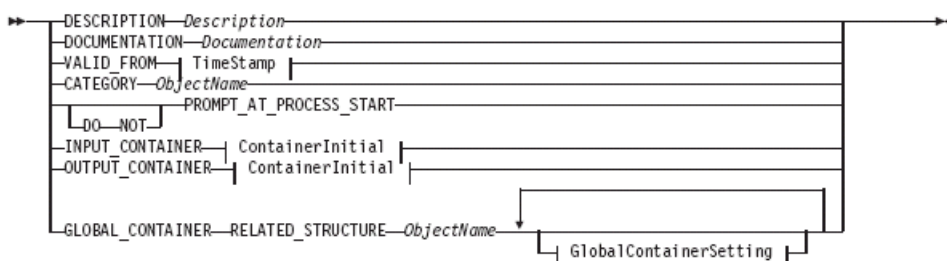
⁸⁵ See "Program execution properties" on page 145.

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 <i>ProcessName</i>	x				
<i>Default data structure</i>	x				
<u>Data structure</u>					113
<u>ProcessSetting</u>					115
<u>DefaultProcessSetting</u>					108
<u>DefaultActivitySetting</u>					110
<u>ProcessStaffAssignmentSetting</u>					116
ProcessGraphicSetting		x ⁸⁶			
<u>Construct</u>					117

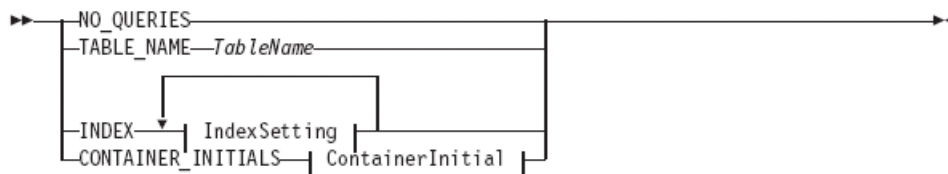
ProcessSetting



⁸⁶ See "Graphical layout information" on page 132.

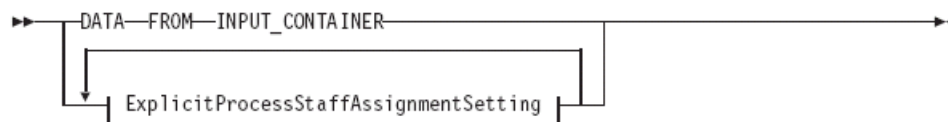
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
DESCRIPTION <i>Description</i>	X				
DOCUMENTATION <i>Documentation</i>	X				
VALID FROM <i>TimeStamp</i>	X				
CATEGORY <i>ObjectName</i>	X ⁸⁷				
PROMPT AT PROCESS START	X				
DO NOT PROMPT AT PROCESS START				X ⁸⁸	
INPUT_CONTAINER <i>ContainerInitial</i>	X				
OUTPUT_CONTAINER <i>ContainerInitial</i>	X				
GLOBAL_CONTAINER RELATED_STRUCTURE <i>ObjectName</i>	X				
GlobalContainerSetting					116

GlobalContainerSetting



FDL syntax expression ⁸⁹	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 <i>TableName</i>				X	
INDEX <i>IndexSetting</i>				X	
CONTAINER_INITIALS <i>ContainerInitial</i>	X				

ProcessStaffAssignmentSetting



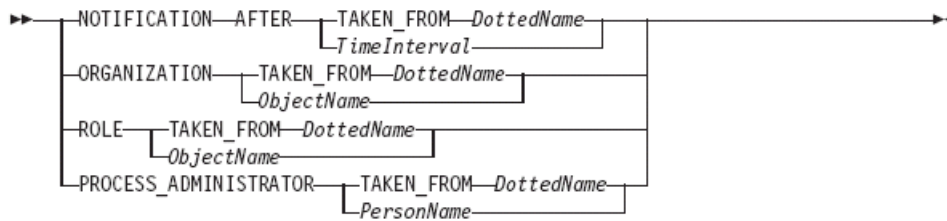
⁸⁷ See "Staff assignment by process category" on page 139.

⁸⁸ See "Prompting for data at process start" on page 134.

⁸⁹ 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 ⁹⁰	
<u>ExplicitProcessStaffAssignmentSetting</u>					117

ExplicitProcessStaffAssignmentSetting



FDL syntax expression ⁹¹	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 <i>DottedName</i>				x	
NOTIFICATION AFTER <i>TimeInterval</i>				x	
ORGANIZATION TAKEN_FROM <i>DottedName</i>	x				
ORGANIZATION <i>ObjectName</i>	x				
ROLE TAKEN_FROM <i>DottedName</i>	x				
ROLE <i>ObjectName</i>	x				
PROCESS_ADMINISTRATOR TAKEN_FROM <i>DottedName</i>	x				
PROCESS_ADMINISTRATOR <i>PersonName</i>	x				

Construct

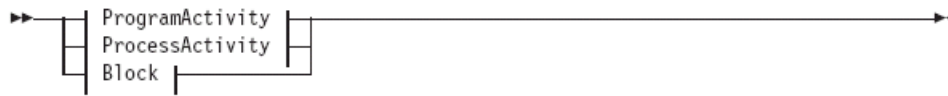


⁹⁰ See "Dynamic staff assignment from input **container**" on page 141.

⁹¹ See "Process notification" on page 139.

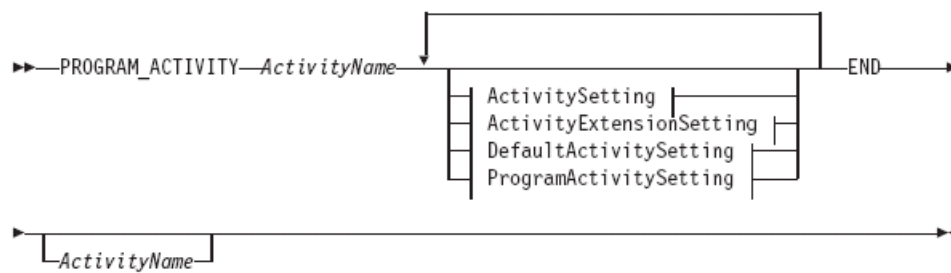
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
<u>Activity</u>					118
<u>ControlFlow</u>					127
<u>DataFlow</u>					127

Activity



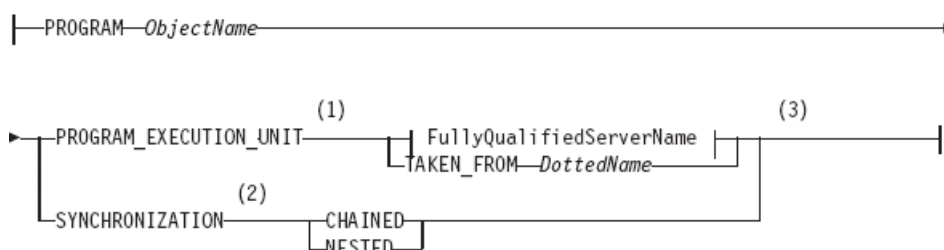
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
<u>ProgramActivity</u>					118
<u>ProcessActivity</u>					120
<u>Block</u>					121

ProgramActivity



FDL syntax expression ⁹²	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 <i>ActivityName</i>	x				
<i>ActivitySetting</i>					122
<i>ActivityExtensionSetting</i>					123
<i>DefaultActivitySetting</i>					110
<i>ProgramActivitySetting</i>					119

ProgramActivitySetting



Notes:

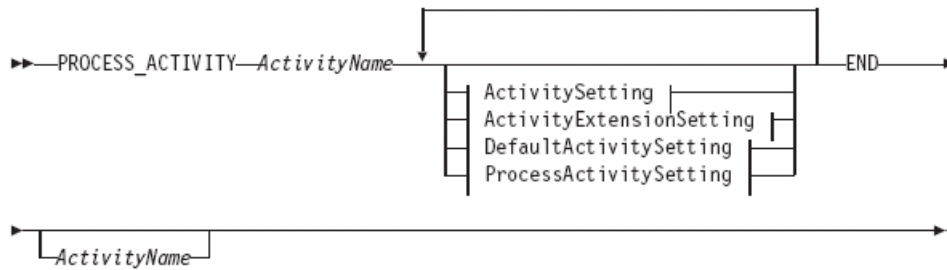
- 1 The keyword `PROGRAM_EXECUTION_SERVER` can still be used instead of `PROGRAM_EXECUTION_UNIT`. However, for new definitions, use only `PROGRAM_EXECUTION_UNIT`, because the old keyword is only valid as an interim solution for this release. The type of server can be a `PROGRAM_EXECUTION_SERVER` or a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

⁹² See "FDL program activity" on page 146.

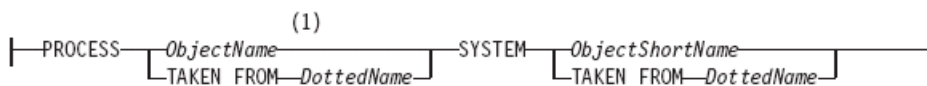
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROGRAM <i>ObjectName</i>	x				
PROGRAM_EXECUTION_UNIT <i>FullyQualifiedServerName</i>	x				
PROGRAM_EXECUTION_UNIT TAKEN_FROM <i>DottedName</i>				x ⁹³	
SYNCHRONIZATION CHAINED	x				
SYNCHRONIZATION NESTED	x				

ProcessActivity



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
PROCESS_ACTIVITY <i>ActivityName</i>	x				
<u>ActivitySetting</u>					122
<u>ActivityExtensionSetting</u>					123
<u>DefaultActivitySetting</u>					110
<u>ProcessActivitySetting</u>					120

ProcessActivitySetting



Notes:

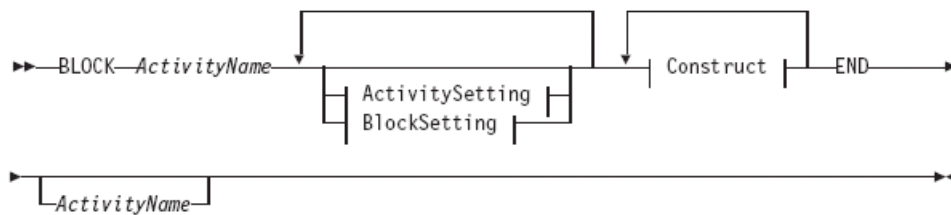
- 1 This is the name of a process.

⁹³ See "Program execution unit taken from container" on page 143.

⁹⁴ See "Staff assignment and notification criteria assigned to process and **UPES activities**" on page 142.

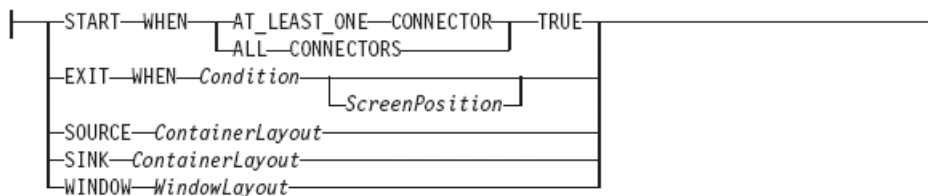
FDL syntax expression ⁹⁵	Migration supported with FDL2BPPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS <i>ObjectName</i>	x				
PROCESS TAKEN_FROM <i>DottedName</i>	x				
SYSTEM <i>ObjectShortName</i>				x	
SYSTEM TAKEN_FROM <i>DottedName</i>				x	

Block



FDL syntax expression	Migration supported with FDL2BPPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
BLOCK <i>ActivityName</i>	x				
<u>ActivitySetting</u>					122
<u>BlockSetting</u>					121
<u>Construct</u>					117

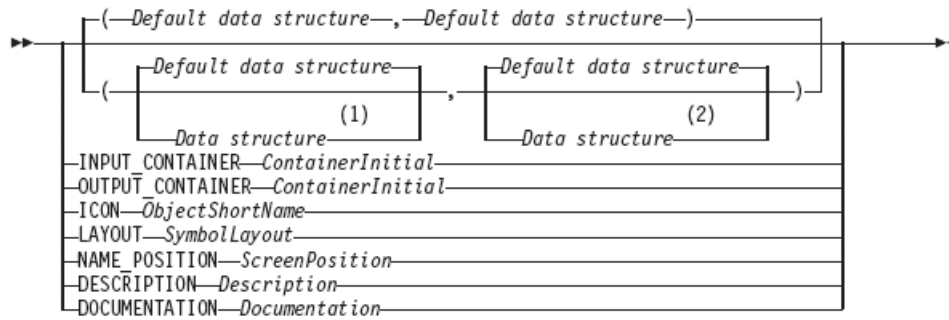
BlockSetting



⁹⁵ See "System properties assigned to process activities" on page 145.

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					129
<i>ScreenPosition</i>		X ⁹⁶			
SOURCE	x				
SINK	x				
<i>ContainerLayout</i>		X ⁹⁶			
WINDOW <i>WindowLayout</i>		X ⁹⁶			

ActivitySetting



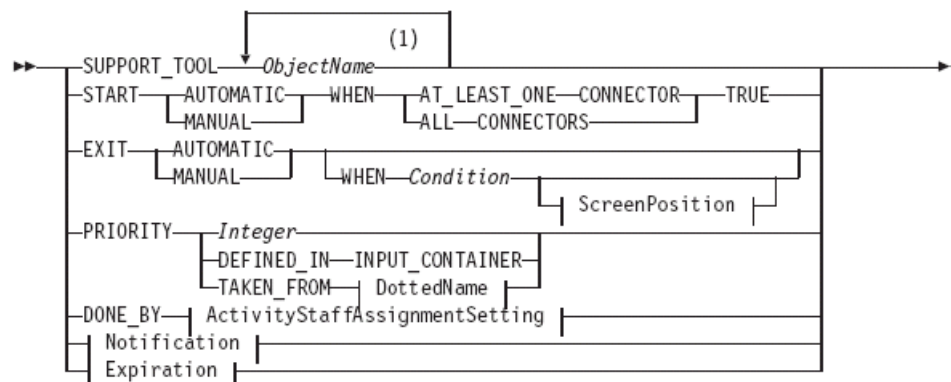
Notes:

- 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
<i>Default data structure</i>	x				
<i>Data structure</i>	x				
INPUT_CONTAINER <i>ContainerInitial</i>	x				
OUTPUT_CONTAINER <i>ContainerInitial</i>	x				
ICON <i>ObjectShortName</i>				X ⁹⁶	
LAYOUT <i>SymbolLayout</i>		X ⁹⁶			
NAME_POSITION <i>ScreenPosition</i>				X ⁹⁶	
DESCRIPTION <i>Description</i>	x				
DOCUMENTATION <i>Documentation</i>	x				

⁹⁶ See "Graphical layout information" on page 132.

ActivityExtensionSetting

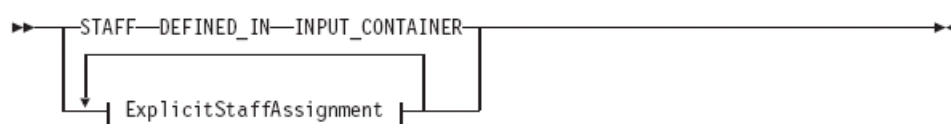


Notes:

- 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 ⁹⁷	
START AUTOMATIC	X ⁹⁸				
START MANUAL		X ⁹⁸			
START ... WHEN AT_LEAST_ONE_CONNECTOR TRUE	X				
START ... WHEN ALL_CONNECTORS TRUE	X				
EXIT AUTOMATIC	X ⁹⁸				
EXIT MANUAL		X ⁹⁸			
EXIT ... WHEN <u>Condition</u>	X				
<u>Condition</u>					129
<u>ScreenPosition</u>				X ⁹⁹	
PRIORITY <u>Integer</u>	X				
PRIORITY DEFINED_IN INPUT_CONTAINER	X				
PRIORITY TAKEN_FROM DottedName	X				
DONE_BY <u>ActivityStaffAssignmentSetting</u>					123
<u>Notification</u>					125
<u>Expiration</u>					126

ActivityStaffAssignmentSetting



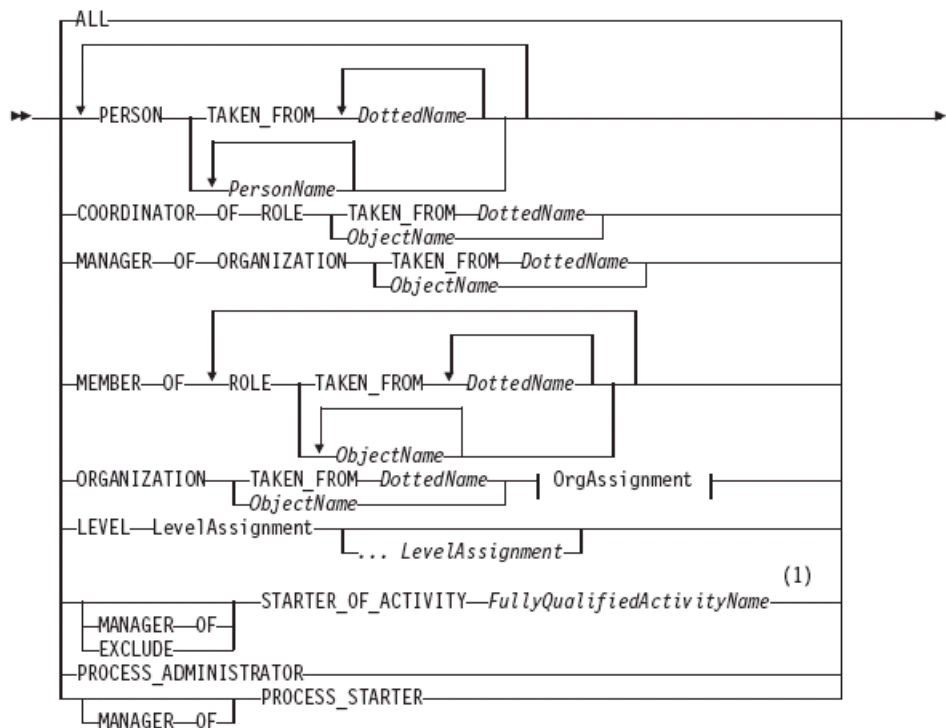
⁹⁷ See "Support tools" on page 147.

⁹⁸ See "Activity start and exit mode" on page 133.

⁹⁹ See "Graphical layout information" on page 132.

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 ¹⁰⁰	
<u>ExplicitStaffAssignment</u>					124

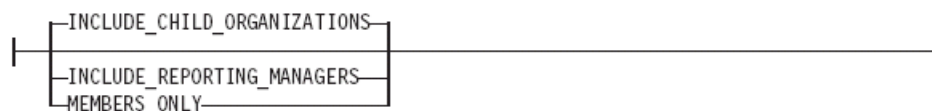
ExplicitStaffAssignment



¹⁰⁰ See "Dynamic staff assignment from input **container**" on page 141.

FDL syntax expression ¹⁰¹	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 <i>DottedName</i>	X				
PERSON <i>PersonName</i>	X				
COORDINATOR OF ROLE TAKEN_FROM <i>DottedName</i>				X	
COORDINATOR OF ROLE <i>ObjectName</i>				X	
MANAGER OF ORGANIZATION TAKEN_FROM <i>DottedName</i>				X	
MANAGER OF ORGANIZATION <i>ObjectName</i>				X	
MEMBER OF ROLE TAKEN_FROM <i>DottedName</i>	X				
MEMBER OF ROLE <i>ObjectName</i>	X				
ORGANIZATION TAKEN_FROM	X				
ORGANIZATION <i>ObjectName</i>	X				
OrgAssignment					125
LEVEL <i>LevelAssignment</i>				X	
MANAGER OF STARTER_OF_ACTIVITY <i>FullyQualifiedActivityName</i>	X				
EXCLUDE STARTER_OF_ACTIVITY <i>FullyQualifiedActivityName</i>	X				
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 ¹⁰¹			
MEMBERS_ONLY	X				

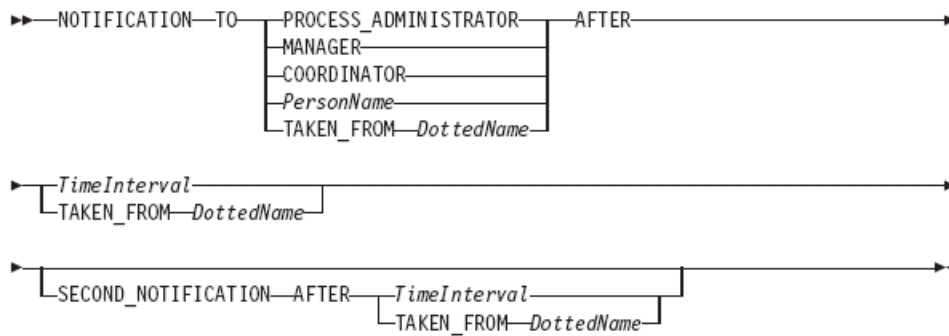
Notification



¹⁰¹ See "Staff assignment criteria" on page 141.

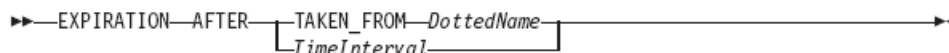
FDL syntax expression ¹⁰²	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
<i>ExplicitNotification</i>					126
DEFINED_IN INPUT_CONTAINER				X ¹⁰³	

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 ¹⁰⁴			
NOTIFICATION TO <i>PersonName</i>	X				
NOTIFICATION TO TAKEN_FROM	X				
NOTIFICATION ... AFTER <i>TimeInterval</i>	X				
NOTIFICATION ... AFTER TAKEN_FROM <i>DottedName</i>				X ¹⁰³	
SECOND_NOTIFICATION AFTER <i>TimeInterval</i>	X				
SECOND_NOTIFICATION AFTER TAKEN_FROM <i>DottedName</i>				X ¹⁰³	

Expiration



¹⁰² See "Notification" on page 138.

¹⁰³ See "Notification from container" on page 139.

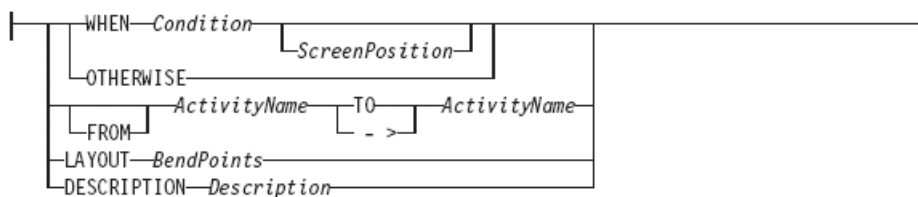
¹⁰⁴ See "Staff assignment criteria" on page 141.

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
EXPIRATION AFTER TAKEN_FROM <i>DottedName</i>	X				
EXPIRATION AFTER <i>TimeInterval</i>	X				

ControlFlow



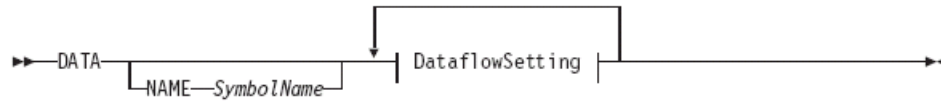
ControlSetting:



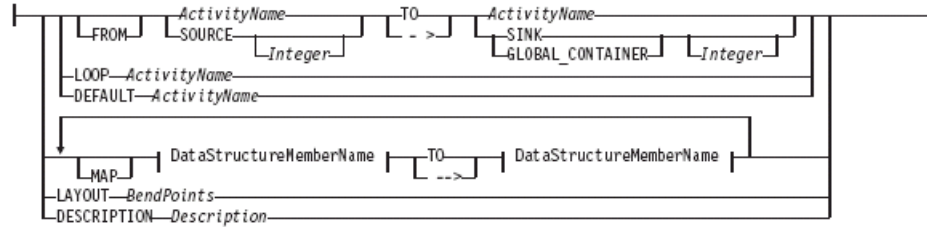
FDL syntax expression	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
CONTROL	X				
NAME <i>SymbolName</i>	X				
WHEN Condition					129
OTHERWISE	X				
FROM? <i>ActivityName</i> (TO ->) <i>ActivityName</i>	X				
LAYOUT <i>BendPoints</i>				X ¹⁰⁵	
DESCRIPTION <i>Description</i>	X				

DataFlow

¹⁰⁵ See "Graphical layout information" on page 132.

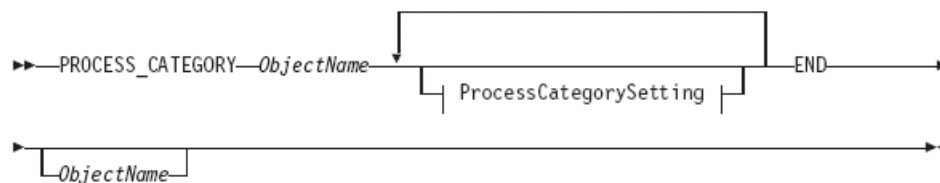


DataflowSetting:



FDL syntax expression	Migration supported with FDL2BPPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
DATA	x				
NAME <i>SymbolName</i>	x				
FROM? (<i>ActivityName</i> SOURCE <i>Integer?</i>) (TO ->) (<i>ActivityName</i> SINK <i>Integer?</i> GLOBAL_CONTAINER <i>Integer?</i>)	x				
LOOP <i>ActivityName</i>	x				
DEFAULT <i>ActivityName</i>	x				
MAP? <i>DataStructureMemberName</i> (TO ->) <i>DataStructureMemberName</i>	x				
LAYOUT <i>BendPoints</i>				x ¹⁰⁶	
DESCRIPTION <i>Description</i>	x				

Process category



ProcessCategorySetting:



¹⁰⁶ See "Graphical layout information" on page 132.

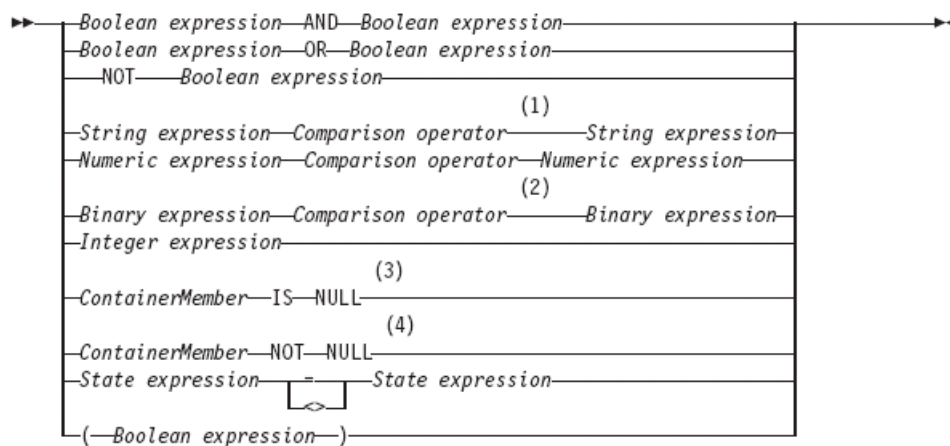
FDL syntax expression ¹⁰⁷	Migration supported with FDL2BPEL Conversion	Workaround possible with manual rework	Outside the scope of process model migration	Not applicable, no workaround possible	See page
PROCESS_CATEGORY <i>ObjectName</i>	X ¹⁰⁸				
DESCRIPTION <i>Description</i>				X	
DOCUMENTATION <i>Description</i>				X	

Conditions

Condition



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.

¹⁰⁷ See "Process category description and documentation" on page 144.

¹⁰⁸ See "Staff assignment by process category" 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
<i>Boolean expression (AND OR) Boolean expression</i>	X				
NOT <i>Boolean expression</i>	X				
<i>String expression Comparison operator String expression</i>	X				
<i>Numeric expression Comparison operator Numeric expression</i>	X				
<i>Binary expression Comparison operator Binary expression</i>		X			
<i>Integer expression</i>	X				
<i>Container member IS NULL</i>	X				
<i>Container member NOT NULL</i>	X				
<i>State expression (= <>) State expression</i>	X ¹⁰⁹				

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
<i>TimeInterval</i>				X ¹¹⁰	
FOREVER	X				
NEVER	X				

¹⁰⁹ See "Activity states" on page 133.

¹¹⁰ See "Policy for how to deal with finished process instances" on page 148.

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.¹¹¹

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 where 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".¹¹¹

Explanation

FDL2BPEL Conversion 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 FDL2BPEL Conversion 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

FDL2BPEL Conversion 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.^{111 112}

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 FDL2BPEL Conversion is not aware of the database system that will store the final BPEL process models. That is why it handles the declaration and delete actions as explained in the following hint:

¹¹¹ See "FDL source file" on page 105.

¹¹² See "DeclarationAction" on page 105.

Hint

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**Description**

FDL2BPEL Conversion 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").

Explanation

While FDL2BPEL Conversion 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.

Hint

In such cases, it is helpful to locate the same message in the generated files using a source editor (usually the files with extension "*.bpe1"). 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****Description**

You cannot migrate a customized WMQWF Buildtime tool set.¹¹³

Explanation

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**Description**

FDL2BPEL Conversion does not migrate any graphical layout properties of a WMQWF business process.^{114 115 116 117 118}

¹¹³ See "DeclarationAction" on page 105.

¹¹⁴ See "Process" on page 115.

¹¹⁵ See "BlockSetting" on page 121.

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**

FDL2BPEL Conversion does not migrate user-defined icons assigned to activity nodes.¹¹⁹

Control flow**Activity start and exit mode****Description**

FDL2BPEL Conversion ignores the FDL activity start and exit modes "manual" and "automatic".¹²⁰

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".¹²¹

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.¹²²

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.

¹¹⁶ See "ActivitySetting" on page 122.

¹¹⁷ See "ControlFlow" on page 127.

¹¹⁸ See "DataFlow" on page 127.

¹¹⁹ See "ActivitySetting" on page 122.

¹²⁰ See "ActivityExtensionSetting" on page 123.

¹²¹ See "Activity classification rules" on page 57.

¹²² See "FDL program activity" on page 146.

Hint

But the migration of references to state "_Expired" in condition expressions is supported (see "Activity expiration" on page 71).

References to binary data in conditions**Description:**

FDL2BPEL Conversion cannot migrate binary condition expressions.

Explanation

FDL2BPEL Conversion translates exit and transition conditions in FDL models to XPath which does not support binary expressions.

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. If you nevertheless depend on using binary expressions in conditions, you can replace the generated XPath condition by a Java condition.

Data flow**Prompting for data at process start****Description**

FDL2BPEL Conversion ignores the process setting "Force prompt for data at process start".¹²³

Explanation

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

Global container database settings**Description**

FDL2BPEL Conversion does not migrate database settings for a global data container.^{124 125} Unsupported global container database settings comprise:

- 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:

¹²³ See "ProcessSetting" on page 115.

¹²⁴ See "GlobalContainerSetting" on page 116.

¹²⁵ See "Queryable global data container" on page 52.

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, FDL2BPEL Conversion converts FDL data structures to corresponding message types that include the predefined data members like "_RC" and "_ACTIVITY_INFO".

Hint

You can run FDL2BPEL Conversion with option **-pi** that requests the initialization of the predefined data members `_ACTIVITY`, `_PROCESS`, and `_PROCESS_MODEL`. Option **-pn** disables the generation of predefined members if you do not need them.¹²⁶

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:

"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 *snippets* with data mapping instructions to the BPEL model for after migration.

Data arrays

Description:

WPS cannot execute Java snippets or "Assign" activities¹²⁷ which assign data values to array elements in non-ascending or interrupted index order.

Explanation:

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

¹²⁶ This option is only available in the commandline mode.

¹²⁷ For instance in the "Assign" activity "Set the data default values" of a business model migrated from WMQWF.

WMQW Buildtime:

The screenshot shows a data connector between two activities, Act1 and Act2. Below the diagram is a dialog box titled "Act1 - Act2 - Data connector <Process DataMapping_with_arrays>". The dialog displays the mapping between the Origin Data Structure (Act1) and the Target Data Structure (Act2).

Origin Data Structure		Target Data Structure		
Member	Type	Member	Type	Mapping
Act1		Act2		
--_RC	LONG	--_PROCE		
--_PROCESS	STRING	--_ACTIV1		
--_PROCESS_MODEL	STRING	--_STRUC	purchaseOrder	
--_ACTIVITY	STRING	Orde (5)		
--_PROCESS_INFO		-- O OrderItem		Act1:OrderItems(0)
--_ACTIVITY_INFO		-- O OrderItem		
--_STRUCT	purchaseOrder	-- O OrderItem		Act1:OrderItems(2)
OrderItems	OrderItem(5)	-- O OrderItem		
OrderItems(0)	OrderItem	-- O OrderItem		
OrderItems(1)	OrderItem	-- O OrderItem		
OrderItems(2)	OrderItem	-- O OrderItem		
OrderItems(3)	OrderItem	-- O OrderItem		
OrderItems(4)	OrderItem	-- O OrderItem		
OrderDate	STRING	Orde	STRING	

The dialog also includes a "Data Mapping" field, buttons for "OK", "Cancel", "Apply", "Reset", and "Help", and a status bar indicating "Object locked by user ID ADMIN" and "Diagram storage mode".

Figure 93: 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:

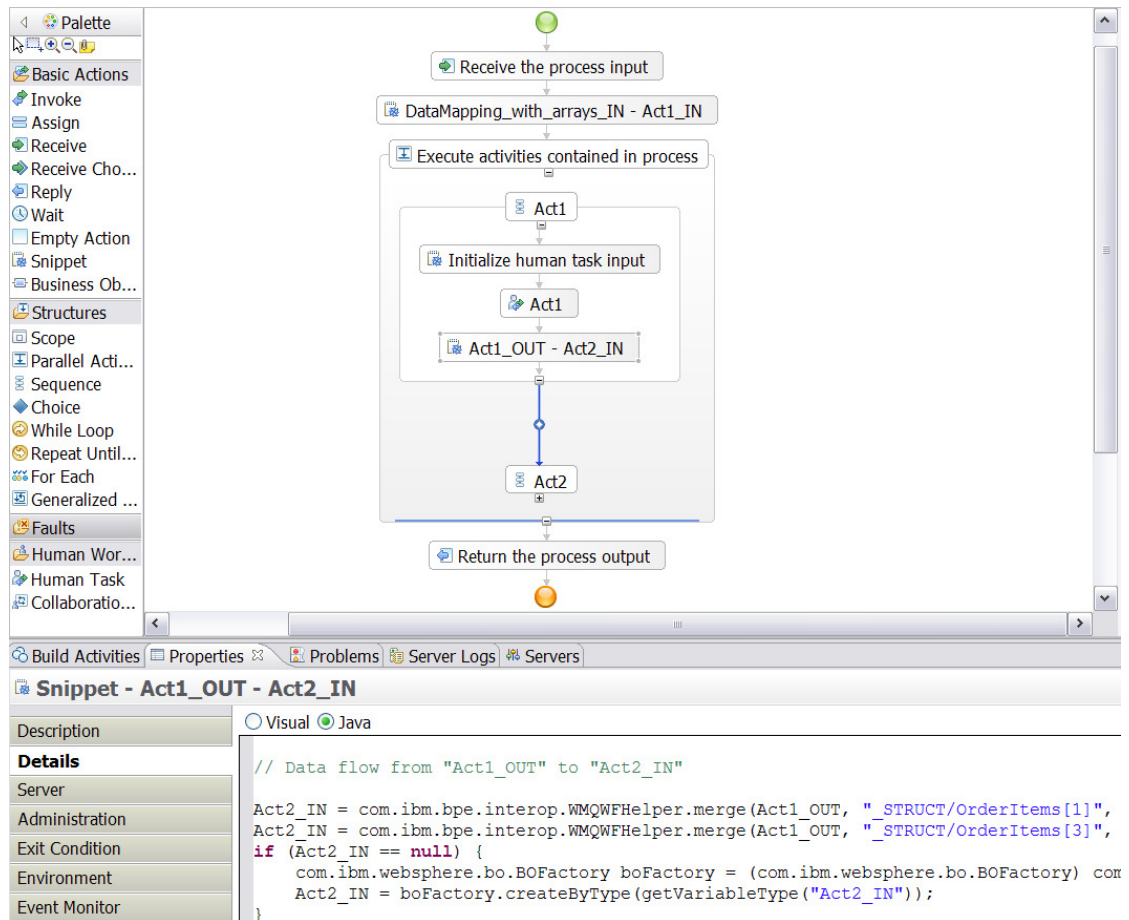


Figure 94: Snippet "Act1_OUT - Act2_IN" contains array mapping

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]"¹²⁸ 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.¹²⁹

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 0.

¹²⁸ Note that FDL expression "OrderItems(1)" corresponds to XPath expression "_STRUCT/OrderItems[2]".

¹²⁹ Note that **position()** returns values > 0, only.

Staff assignment and notification

WMQWF staff repository

Description

FDL2BPEL Conversion ignores any staff definitions that may be contained in your FDL file (for example, "Person", "Role", "Organization", and "Level" definitions).¹³⁰

Explanation

The target artifacts of FDL2BPEL Conversion 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.¹³¹

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 57).^{132 133}

Notification mode

Description

FDL2BPEL Conversion does not support the migration of notification mode settings.^{133 134 135}

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.¹³⁶ There is no escalation mode available that is equivalent to "HOLD". If the task instance is suspended, the escalation timer keeps running.

Notification policy

Description

FDL2BPEL Conversion does not support the migration of notification policy settings "Assign substitute for notification if user is absent" and "Send second notification to same user".¹³⁷

Explanation

¹³⁰ See "DeclarationAction" on page 105.

¹³¹ See "MQ Workflow LDAP Bridge" on page 31 in IBM WebSphere MQ Workflow "Administration Guide", Version 3.6, SH12-6289

¹³² See "ExplicitProcessStaffAssignmentSetting" on page 117.

¹³³ See "Notification" on page 125.

¹³⁴ See "DefaultProcessSetting" on page 108.

¹³⁵ See "ExplicitNotification" on page 126.

¹³⁶ See "Notification" on page 83.

¹³⁷ See "DefaultActivitySetting" on page 110.

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

Notification from container

Description

FDL2BPEL Conversion does not support the migration of dynamically assigned notification settings in the input container.¹³⁸

Process notification

Description

WMQWF can be set up to notify a person if a specified maximum duration of a process instance is exceeded. FDL2BPEL Conversion does not support the migration of notification settings on the process level.¹³⁹

Explanation

There is no equivalent notification function available in WPS.

Staff assignment by process category

Description

FDL2BPEL Conversion supports migrating the process category property. However, other than WMQWF, WPS does not support staff assignment by process category.^{140 141}

Hint

Possible migration solution:

1. Introduce a new group that corresponds to the FDL process category.
2. Users previously authorized for the category must be reassigned to the new role (group).
3. Create an human (invocation) task for the potential starters of your process

Figure 95 and Figure 96 indicate how you can execute step 3:

In the WID process editor diagram select the properties of activity "Receive the process input" -> click on page "Authorization" -> click on **New...** -> the task editor opens -> change the people assignment criteria of the potential starters to "Group Members" -> assign the process category name to the new group name and set the property "IncludeSubgroups" according to your needs.

¹³⁸ See "Notification" on page 83.

¹³⁹ See "ExplicitProcessStaffAssignmentSetting" on page 117.

¹⁴⁰ See "ProcessSetting" on page 115.

¹⁴¹ See "Process category" on page 128.

The screenshot displays a BPMN editor interface. On the left is a 'Palette' with categories like 'Basic Actions' and 'Structures'. The main workspace shows a process flow starting with a green start node, followed by tasks: 'Receive the process input', 'Set the data default values', 'Add default values to the process input', and 'Distribute the process input data to the start activities'. A large container task 'Execute activities contained in process' contains a 'Validate' task, which branches into 'SendEmailAck' and 'TerminateProcess'. 'SendEmailAck' further branches into 'ReserveHotel', 'BookCar', and 'ReserveFlight'. Below the workspace is a toolbar with 'Build Activities', 'Properties', 'Problems', 'Server Logs', and 'Servers'. The 'Properties' panel for the selected 'Receive - Receive the process input' task is visible, showing a description and a 'Human Task' field set to '-- None --'. A red circle highlights the 'New...' button next to the 'Human Task' field, with a mouse cursor pointing at it.

Receive - Receive the process input

Description: Potential starters of the process can be specified using a human task.

Human Task: -- None -- **New...** Remove

Authorization

Figure 95: Creating a human task for the potential starters of the migrated process

Receive_the_process_inputInvocationTask

▼ Invocation Task

Name Receive_the_process_inputInvocationTask

▼ Service Interface

▼ People Assignment (Originator) + ×

Potential Starters Group Members

GroupName *	Travel
IncludeSubgroups *	*

▼ User Interface + ×

User Interface

▼ Escalation

Running

Build Activities Properties Problems Server Logs Servers

► Staff role: Potential Starters

Assign People People assignment criteria: Group Members

Assigns members of groups.
Use this to create individual assignments for every group member.
Define a group name as a uniqueName.

Name	Value
GroupName *	Travel
IncludeSubgroups *	*
Domain	
AlternativeGroupName1	
AlternativeGroupName2	

Figure 96: Specifying the potential starters as group members

Dynamic staff assignment from input container

Description

Staff assignment criteria taken from the predefined data members in the input container is limited to data member **Organization**.^{142 143}

Staff assignment criteria

Description

¹⁴² See "ProcessStaffAssignmentSetting" on page 116.

¹⁴³ See "ActivityStaffAssignmentSetting" on page 123.

FDL2BPEL Conversion cannot map all staff assignment criteria that you specified for a WMQWF process or for program activities that are contained in it.^{144 145 146}

Explanation

WMQWF and the BPC Human Task Manager support different user registry systems and the staff query languages. Table 19 in section "Mapping WMQWF staff assignment criteria to TEL people assignment criteria", on page 83, 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 like "Coordinator of role".

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 and UPES activities

Description

FDL2BPEL Conversion ignores all staff assignment and notification criteria that you specified for a process activity.¹⁴⁷ The same is true for a program activity that FDL2BPEL Conversion classifies as a "Service invocation activity" (UPES activity).¹⁴⁸

Explanation

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

Migration of "Exclude starter of activity"

Description

The migration of WMQWF staff assignment criterion "Exclude starter of activity <*name of remote activity*>" (aka. "4 eyes principle") to a corresponding TEL people assignment criterion is limited to the following staff assignment criteria of the remote activity:

- People (mapped to "Users by user ID")
- Members of roles (mapped to "Group Members")
- Organization (mapped to "Group Members")
- Exclude starter of activity

The corresponding TEL people assignment criterion is accordingly either "Users by user ID without Named users" or "Group Members without Named users".

Explanation

"Users by user ID without Named users" and "Group Members without Named users" are the only TEL people assignment criteria which allow for excluding people from staff resolution.

Hint

Note that cascading "Exclude starter of activity" along the chain of several activities corresponds to the "2n eyes principle" in WMQWF, but its migration to "Users by user ID without Named users" or "Group Members without Named users" will behave differently in WPS, be-

¹⁴⁴ See "DefaultActivitySetting" on page 110.

¹⁴⁵ See "ExplicitStaffAssignment" on page 124.

¹⁴⁶ See "OrgAssignment" on page 125.

¹⁴⁷ See "ProcessActivity" on page 120.

¹⁴⁸ See "Activity classification rules" on page 57.

cause only the starter of the immediately preceding activity will be excluded from staff resolution.

Program execution server

User-defined program execution server (UPES)

Description

FDL2BPEL Conversion does not migrate FDL server definitions, except for user-defined program execution server (UPES).^{149 150}

Explanation

FDL2BPEL Conversion 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 95.

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.

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

FDL2BPEL Conversion migrates WMQWF program activities that invoke a UPES, provided that the UPES application does not depend on XML message types other than:¹⁴⁹

• 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 95.

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

¹⁴⁹ See "Server" on page 110.

¹⁵⁰ See "UPESContext" on page 112.

FDL2BPEL Conversion 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

FDL2BPEL Conversion does not migrate the data structure properties "description" and "documentation".¹⁵¹

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

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

Process category description and documentation

Description

FDL2BPEL Conversion does not migrate a description and documentation that you assigned to a process category.¹⁵²

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.¹⁵³

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 which it belongs to.¹⁵⁴

¹⁵¹ See "Data structure" on page 113.

¹⁵² See "Process category" on page 128.

¹⁵³ See "Topology" on page 106.

¹⁵⁴ See "TopologySetting" on page 107.

Audit settings

Description

The migration of audit event definitions and settings of the WMQWF auditing system is not supported.^{155 156}

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 FDL2BPEL Conversion uses the FDL properties "AUDIT_FILTER_DB" and "AUDIT_FILTER_DB" as criteria to set the "business relevant" property in BPEL to "yes".^{157 158}

Process autonomy modes

Description

You cannot migrate process autonomy modes "full", "staff", "notification", and "administration".¹⁵⁹

Explanation

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

Hint

FDL2BPEL Conversion 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".¹⁶⁰

Program execution properties

Description

FDL2BPEL Conversion does not migrate the program execution properties of an FDL program definition.¹⁶¹

Explanation

FDL2BPEL Conversion uses FDL program definitions to generate corresponding interface definitions (WSDL "port types"¹⁶²) 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

¹⁵⁵ See "DefaultProcessSetting" on page 108.

¹⁵⁶ See "DefaultActivitySetting" on page 110.

¹⁵⁷ See "Process property "business relevant"" on page 49.

¹⁵⁸ See "Activity property "business relevant"" on page 63.

¹⁵⁹ See "Autonomy" on page 109.

¹⁶⁰ See "Process" on page 115.

¹⁶¹ See "Program" on page 114.

¹⁶² See "Mapping FDL PROGRAM to a port type" on page 44.

FDL2BPEL Conversion ignores FDL properties of a process activity that determine on which system a subprocess should run.¹⁶³

Model accuracy

FDL program activity

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.¹⁶⁴

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

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

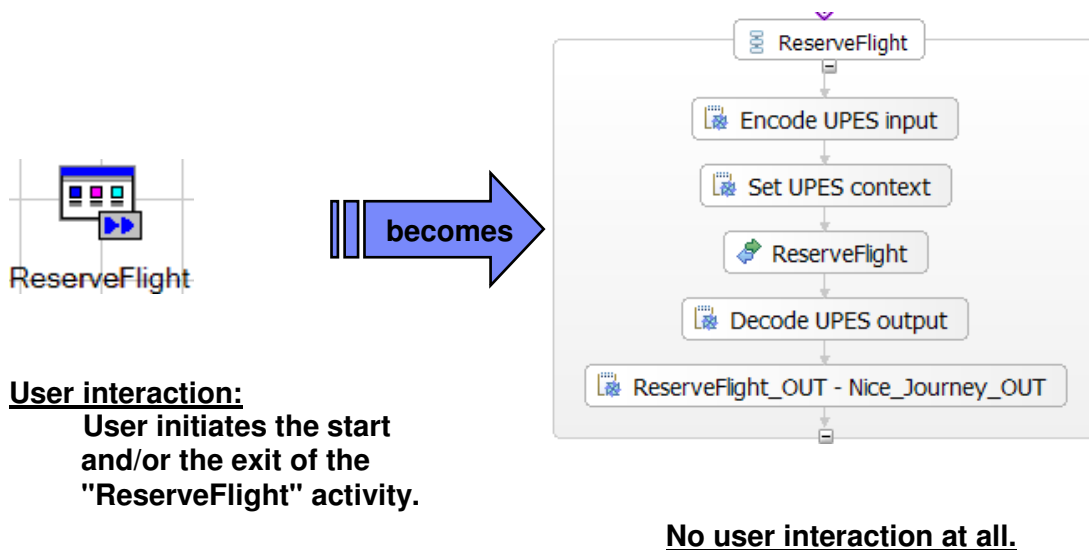


Figure 97: User interaction is not translated to BPEL

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

¹⁶³ See "ProcessActivitySetting" on page 120.

¹⁶⁴ See "ProgramActivity" on page 118.

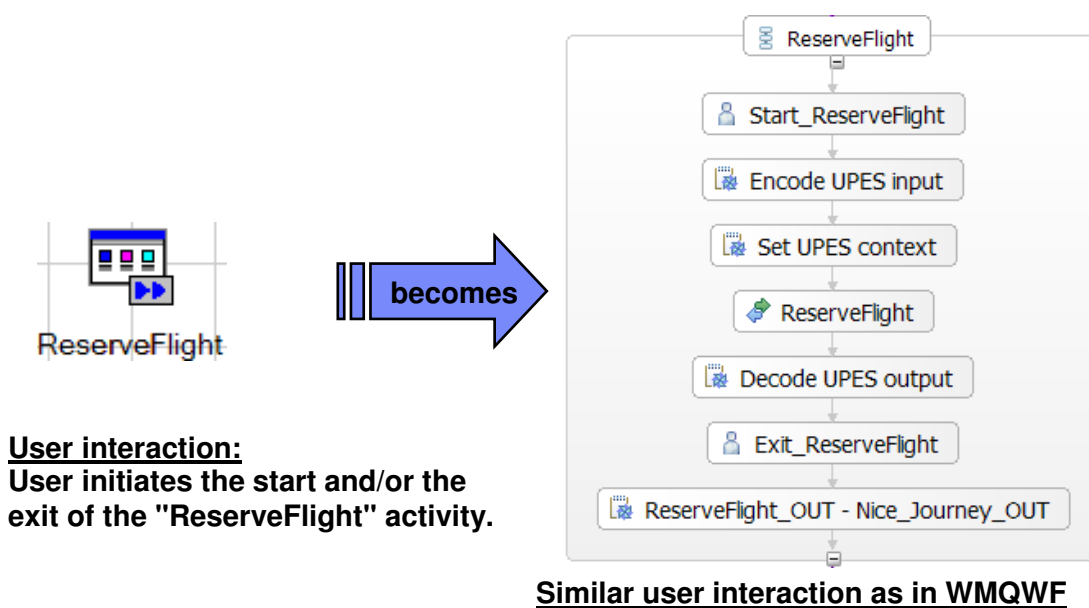


Figure 98: Inserting "Human Task" activities for manual activity start and exit

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. FDL2BPEL Conversion ignores the support tools specified in the FDL input file.¹⁶⁵
166

Explanation

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

Workitem refresh policy

Description

FDL2BPEL Conversion does not support the migration of a refresh policy.¹⁶⁷

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.

Note that the refreshing behavior of the BPC explorer is equivalent to the "PULL"-policy.

Policy for how to deal with finished workitems

Description

¹⁶⁵ See "DeclarationAction" on page 105.

¹⁶⁶ See "ActivityExtensionSetting" on page 123.

¹⁶⁷ See "DefaultProcessSetting" on page 108.

FDL2BPEL Conversion does not support the WMQWF policy for how to deal with finished workitems.¹⁶⁷

Policy for how to deal with finished process instances

Description

FDL2BPEL Conversion does not support the specification of a time period for the policy for how long finished process instances should be kept.^{167 168}

Hint

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

¹⁶⁸ See "TimePeriod" on page 130.