



*The Workflow Management Coalition Specification*

# Workflow Management Coalition Workflow Standard

## Process Definition Interface -- XML Process Definition Language

Document Number WFMC-TC-1025  
Document Status –Released

August 30, 2012  
Version 2.2

Copyright © 2012 The Workflow Management Coalition

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the Workflow Management Coalition except that reproduction, storage or transmission without permission is permitted if all copies of the publication (or portions thereof) produced thereby contain a notice that the Workflow Management Coalition and its members are the owners of the copyright therein.

wfmc@wfmc.org  
<http://www.wfmc.org>

The “WfMC” logo and “Workflow Management Coalition” name are service marks of the Workflow Management Coalition. Neither the Workflow Management Coalition nor any of its members make any warranty of any kind whatsoever, express or implied, with respect to the Specification, including as to non-infringement, merchantability or fitness for a particular purpose. This Specification is provided “as is”.

# Table of Content

1.1. ACKNOWLEDGEMENTS .....	7
<b>2. AUDIENCE.....</b>	<b>7</b>
<b>3. PURPOSE.....</b>	<b>8</b>
<b>4. INTRODUCTION .....</b>	<b>8</b>
4.1. CONFORMANCE .....	9
4.2. XPDL VERSION COMPATIBILITY .....	9
4.3. REFERENCES .....	9
<b>5. OVERVIEW OF PROCESS DEFINITION INTERCHANGE.....</b>	<b>10</b>
5.1. APPROACHES TO PROCESS DEFINITION INTERCHANGE .....	10
<b>6. META-MODEL .....</b>	<b>12</b>
6.1. PROCESSES AND PACKAGES.....	12
6.2. PACKAGE META-MODEL .....	14
6.2.1. <i>Process Repository</i> .....	15
6.2.1.1. <i>Redefinition and Scope</i> .....	15
6.3. PROCESS META-MODEL .....	16
6.4. ENTITIES OVERVIEW .....	17
6.4.1. <i>Swimlanes</i> .....	17
6.4.1.1. <i>Pool</i> .....	17
6.4.1.2. <i>Lane</i> .....	17
6.4.2. <i>Process Definition</i> .....	18
6.4.3. <i>Process Activity</i> .....	18
6.4.4. <i>Transition Information</i> .....	19
6.4.5. <i>Participant Declaration</i> .....	19
6.4.6. <i>Application Declaration</i> .....	19
6.4.7. <i>Artifact</i> .....	20
6.4.8. <i>Message Flow</i> .....	20
6.4.9. <i>Association</i> .....	21
6.4.10. <i>Relevant Data Field</i> .....	21
6.4.11. <i>Data Types and Expressions</i> .....	21
6.4.12. <i>System and Environmental Data</i> .....	21
6.4.13. <i>Resource Repository</i> .....	21
6.4.14. <i>Vendor or User specific Extensions</i> .....	21
6.4.14.1. <i>Extended Elements and Attributes</i> .....	21
6.4.14.2. <i>Extended parameter mapping</i> .....	21
<b>7. XML PROCESS DEFINITION LANGUAGE.....</b>	<b>23</b>
7.1. ELEMENTS COMMON FOR MULTIPLE ENTITIES .....	23
7.1.1. <i>Id Attribute of Many Objects</i> .....	23
7.1.1.1. <i>Graphic Information</i> .....	23
7.1.1.2. <i>Pages</i> .....	23
7.1.1.3. <i>Coordinate Information</i> .....	23
7.1.1.4. <i>NodeGraphicsInfo</i> .....	24
7.1.1.5. <i>ConnectorGraphicsInfo</i> .....	24
7.1.2. <i>Expression Type</i> .....	25
7.1.3. <i>Extended Attributes</i> .....	25
7.1.3.1. <i>Anonymous Extended Attribute</i> .....	26
7.1.3.2. <i>Namespace Qualified Extensions</i> .....	26
7.1.4. <i>Formal Parameters</i> .....	26
7.1.4.1. <i>Parameter passing semantics</i> .....	27
7.1.4.2. <i>Concurrency semantics</i> .....	27
7.1.5. <i>External Reference</i> .....	27
7.1.5.1. <i>Web Services</i> .....	28

7.1.6.	<i>Actual Parameters</i> .....	28
7.1.6.1.	Formal-actual Parameter Mapping.....	28
7.1.7.	<i>Assignment</i> .....	29
7.1.8.	<i>Category</i> .....	29
7.1.9.	<i>Artifact</i> .....	30
7.1.9.1.	Sequence Flow Connections for Artifacts.....	30
7.1.9.2.	Message Flow Connections for Artifacts .....	30
7.1.9.3.	Schema for Artifacts .....	30
7.1.9.4.	Object.....	31
7.1.9.5.	DataObject in BPMN1.x .....	31
7.1.9.6.	Group .....	32
7.1.10.	<i>Data Object for BPMN2.0</i> .....	33
7.1.10.1.	Data Modeling .....	33
7.1.10.2.	Data Object .....	34
7.1.10.3.	Data Store .....	35
7.1.10.4.	Data Store Reference .....	35
7.1.10.5.	Data Input .....	36
7.1.10.6.	Data Output.....	37
7.1.10.7.	Data Associations.....	37
7.2.	PACKAGE DEFINITION .....	38
7.2.1.	<i>Package definition Header</i> .....	40
7.2.1.1.	Vendor extensions.....	41
7.2.1.2.	Example of specifying the vendor extension's schema location .....	41
7.2.2.	<i>Redefinable Header</i> .....	42
7.2.3.	<i>Conformance Class Declaration</i> .....	42
7.2.3.1.	Graph Conformance .....	42
7.2.3.2.	BPMN Model Portability Conformance .....	42
7.2.3.3.	Conformance Schema .....	44
7.2.3.4.	BPMN2.0 Conformance Subclasses .....	45
7.2.3.5.	Examples.....	49
7.2.4.	<i>Script</i> .....	50
7.2.5.	<i>External Package</i> .....	50
7.2.6.	<i>Example use of External Package</i> .....	51
7.3.	APPLICATION DECLARATION .....	51
7.3.1.	<i>Invocation Parameters</i> .....	52
7.3.2.	<i>Application Types</i> .....	52
7.3.2.1.	EJB.....	52
7.3.2.2.	POJO.....	52
7.3.2.3.	XSLT .....	53
7.3.2.4.	Script.....	53
7.3.2.5.	WebService.....	53
7.3.2.6.	BusinessRule.....	53
7.3.2.7.	Form.....	53
7.4.	SWIMLANES .....	54
7.4.1.	<i>Pool</i> .....	54
7.4.1.1.	BPMN Graphics for Pools .....	54
7.4.1.2.	Schema for Pools .....	56
7.4.2.	<i>Lane</i> .....	57
7.4.2.1.	BPMN Graphics for Lanes.....	57
7.4.2.2.	Schema for Lanes.....	58
7.5.	PROCESS DEFINITION.....	59
7.5.1.	<i>Schema</i> .....	59
7.5.2.	<i>Process Definition Header</i> .....	61
7.5.3.	<i>Process Redefinable Header</i> .....	62
7.5.4.	<i>Activity Set/Embedded SubProcess</i> .....	63
7.5.4.1.	Event Sub-Process .....	64
7.5.5.	<i>ProcessType in BPMN mapping to WS-BPEL</i> .....	66
7.5.6.	<i>Status</i> .....	66
7.5.7.	<i>SuppressJoinFailure</i> .....	67
7.5.8.	<i>EnableInstanceCompensation</i> .....	67

7.5.9.	<i>AdHoc</i> .....	67
7.6.	PROCESS ACTIVITY .....	67
7.6.1.	<i>Execution Control Attributes</i> .....	71
7.6.2.	<i>Route Activity</i> .....	71
7.6.2.1.	Gateway Activity .....	72
7.6.2.2.	Examples of Gateways and their Representation .....	73
7.6.2.3.	BPMN Semantics for Gateway Types .....	74
7.6.3.	<i>Block Activity/Embedded Sub-Process</i> .....	85
7.6.3.1.	Schema for BlockActivity .....	86
7.6.4.	<i>Event Activity</i> .....	86
7.6.4.1.	Schema for Event .....	86
7.6.4.2.	Start Event .....	87
7.6.4.3.	Intermediate Event .....	92
7.6.4.4.	End Event .....	98
7.6.4.5.	Common Elements Used in Start, Intermediate and End Events .....	101
7.6.4.6.	Examples of Events and their representation .....	105
7.6.4.7.	Review of Events .....	105
7.6.5.	<i>Implementation Alternatives</i> .....	107
7.6.5.1.	No Implementation .....	108
7.6.5.2.	Tool .....	108
7.6.5.3.	Task .....	108
7.6.5.4.	SubFlow/Sub-Process .....	114
7.6.5.5.	Reference .....	121
7.6.5.6.	GlobalActivityReference .....	121
7.6.6.	<i>Performer Relationship</i> .....	121
7.6.7.	<i>Deadline</i> .....	122
7.6.7.1.	Schema for Deadline .....	123
7.6.8.	<i>Simulation Information</i> .....	123
7.6.9.	<i>Transition Restriction</i> .....	124
7.6.9.1.	Join .....	124
7.6.9.2.	Split .....	125
7.6.9.3.	BPMN View of Routing Logic .....	126
7.6.10.	<i>InputSets</i> .....	130
7.6.11.	<i>OutputSets</i> .....	130
7.6.12.	<i>Transaction</i> .....	131
7.6.13.	<i>Loop</i> .....	131
7.7.	TRANSITION INFORMATION .....	133
7.7.1.	<i>Condition</i> .....	134
7.7.1.1.	Exception Conditions .....	134
7.7.2.	<i>BPMN view of Transition --- Sequence Flow</i> .....	134
7.7.2.1.	Uncontrolled flow .....	135
7.7.2.2.	Conditional flow .....	135
7.7.2.3.	Default flow .....	135
7.7.2.4.	Exception flow .....	135
7.7.2.5.	Compensation Association .....	136
7.7.2.6.	Sequence Flow Rules .....	136
7.7.2.7.	SequenceFlow Examples .....	137
7.8.	PARTNER LINKS .....	138
7.8.1.	<i>Partner Link Type</i> .....	138
7.8.2.	<i>Partner Link</i> .....	138
7.9.	MESSAGING .....	139
7.9.1.	<i>Message Flow</i> .....	139
7.9.2.	<i>BPMN Graphics and Semantics for Message Flow</i> .....	139
7.9.3.	<i>Schema for Message Flow</i> .....	140
7.9.4.	<i>Message Type</i> .....	140
7.9.5.	<i>End Point</i> .....	141
7.9.6.	<i>Web ServiceOperation</i> .....	141
7.9.7.	<i>Web Service Fault Catch</i> .....	142
7.9.8.	<i>Message Flow Rules</i> .....	142
7.10.	ASSOCIATION .....	143

7.10.1.	<i>BPMN Graphics and Semantics</i> .....	143
7.10.2.	<i>Schema for Association</i> .....	144
7.11.	PARTICIPANTS .....	144
7.11.1.	<i>Participant Entity Types</i> .....	145
7.12.	RELEVANT DATA FIELD/PROPERTY .....	145
7.13.	DATA TYPES.....	146
7.13.1.	<i>Basic Data Types</i> .....	146
7.13.2.	<i>Complex Data Types</i> .....	147
7.13.2.1.	Schema Type.....	147
7.13.2.2.	Record Type.....	147
7.13.2.3.	Union Type .....	148
7.13.2.4.	Enumeration Type.....	148
7.13.2.5.	Array Type.....	148
7.13.2.6.	List Type.....	148
7.13.3.	<i>Declared Data Types</i> .....	148
7.13.3.1.	Type Declaration.....	148
7.13.3.2.	Declared Type .....	149
8.	<b>XPDL SCHEMA</b> .....	<b>150</b>
9.	<b>FIGURES AND TABLES</b> .....	<b>151</b>
9.1.	FIGURES.....	151
9.2.	TABLES.....	153

## 1.1. Acknowledgements

XDPL2.2 was a collaborative effort led by Robert Shapiro and Denis Gagne. Tim Stephenson contributed to the development of the specification and in particular to the area concerning Data Objects. Document revisions were also provided by Justin Brunt.

Denis Gagne and the Trisotech team took responsibility for editing the XPDL schema to incorporate changes and extensions associated with BPMN2.0.

Jay Cousins (Rivcom Associates) contributed to the editing of the specification.

XPDL2.1 was a collaborative effort with contributions from many individuals.

Robert Shapiro (Process Analytica, Cape Visions, Global360) directed the effort and edited the specification.

Keith Swenson (Fujitsu Software) contributed key ideas and as chair of the WfMC Technical Committee helped locate other resources to get this job done.

Justin Brunt (TIBCO Software) organized other resources at TIBCO Software and identified the changes to BPMN introduced in BPMN 1.1. Justin helped push the evolution of BPMN Model Portability.

Bruce Silver (Bruce Silver Associates) fixed a number of BPMN-related problems and helped focus the specification on BPMN diagram interchange. He developed the concept of BPMN Model Portability conformance classes.

Other members of the Global 360 team included Tom Laverty and Andy Adler. Tom did a lot of work on the schema and added referential integrity checking.

Other members of the TIBCO Software team included Tim Stephenson, Sid Allway, Ravikanth Somayaji and Kamlesh Upadhyaya. Tim helped push the evolution of BPMN Model Portability. Sid rewrote the documentation for Route Activity/Gateway and TransitionRestrictions.

Denis Gagné [dgagne@trisotech.com] and his team at Trisotech contributed a spreadsheet showing the differences between BPMN 1.0, BPMN1.1 and XPDL 2.0. They also suggested several enhancements to the schema.

Shane C. Gabie (UNISYS) contributed change proposals.

Brad Stone (Aspirin Software) helped improve the specification.

The BPMN 1.1 Draft with mark-up DTC-2007-06-02 provided detailed information, tables and figures for this specification. Much appreciation goes to the BPMN group in OMG, all of the individuals who participated and especially to Stephen White (IBM).

XPDL2.0 required many hours of work by individuals who had to find time to contribute while carrying out their normal duties for the companies that employ them.

Robert Shapiro (Global 360) and Mike Marin (FileNET) did the bulk of the work.

Justin Brunt, Wojciech Zurek, Tim Stephenson (TIBCO Software), Sasa Bojanic (Prozone) and Gangadhar Gouri (Fujitsu Software) made significant contributions.

Keith Swenson (Fujitsu Software) provided invaluable organizational support and encouragement.

## 2. Audience

The intended audience for this document is primarily vendor organizations who seek to implement the XML Process Definition Language (XPDL) of the Workflow Management Coalition (WfMC), or wish to use it as a file format for the Business Process Modeling Notation (BPMN) of the Business Process Management Initiative (BPMI) and the Object Management Group (OMG). It may also be of interest to those seeking to assess conformance claims made by vendors for their products. Comments should be addressed to the Workflow Management Coalition.

### 3. Purpose

XPDL Version 2.2 is backward compatible with prior versions of XPDL and can be used as a file format for BPMN 2.0 and BPMN 1.x. The original purpose of XPDL is maintained and enhanced by this new version of the specification. The XPDL and the BPMN specifications address the same modeling problem from different perspectives. XPDL provides an XML file format that can be used to interchange process models between tools. BPMN provides a graphical notation to facilitate human communication between business users and technical users, of complex business processes. BPMN 2.0 which was completed in January 2011 introduced its own serialization which is not backward compatible.

There are a number of elements that are present in BPMN version 2.0 that were not present in prior versions. These have been incorporated into this version of XPDL. There are also some changes which have been incorporated without rendering XML documents in prior versions obsolete.

XPDL2.2 focuses on Process Modeling. As such it does not include all parts of the BPMN2.0 specification. In particular, Choreography Diagrams and Conversation Diagrams are not covered. In addition, Collaboration Diagrams are covered only to the extent required to support modeling message flow between Pools.

This document has been expanded to provide much more documentation from the BPMN perspective. It should be possible for BPMN users to draw diagrams and specify attributes for those diagrams based on the contents of this specification.

The WfMC has identified five functional interfaces to a process or workflow service as part of its standardization program. This specification forms part of the documentation relating to "Interface one" - supporting Process Definition Import and Export. This interface includes a common meta-model for describing the process definition (this specification) and also a companion XML schema for the interchange of process definitions.

### 4. Introduction

A variety of different tools may be used to analyse, model, describe and document a business process. The process definition interface defines a common interchange format, which supports the transfer of process definitions between separate products.

The interface also defines a formal separation between the development and run-time environments, enabling a process definition, generated by one modelling tool, to be used as input to a number of different run-time products.

A process definition, generated by a build-time tool, is capable of interpretation in different run-time products. Process definitions transferred between these products or stored in a separate repository are accessible via the XPDL common interchange format.

To provide a common method to access and describe process definitions, a process definition meta-data model has been established. This meta-data model identifies commonly used entities within a process definition. A variety of attributes describe the characteristics of this limited set of entities. Based on this model, vendor specific tools can transfer models via a common exchange format.

One of the key elements of XPDL is its extensibility to handle information used by a variety of different tools. XPDL may never be capable of supporting all additional information requirements in all tools. Based upon a limited number of entities that describe a process definition (the "Minimum Meta Model"), XPDL supports a number of differing approaches.

One of the most important elements of XPDL is a generic construct that supports vendor specific attributes for use within the common representation. We recommend that any missing attributes be proposed to the WfMC for inclusion in future releases.

This document describes the meta-model, which is used to define the objects and attributes contained within a process definition. The XPDL grammar is directly related to these objects and attributes. This approach needs two operations to be provided by a vendor:

- Import a process definition from XPDL.
- Export a process definition from the vendor's internal representation to XPDL.

A vendor can use an XSL style sheet to accomplish these two operations.

All keywords and terms used within this specification are based upon the WfMC Glossary, or terminology used by



BPMN.

For the purpose of this document, the terms process definition, business process model, and workflow model are all considered to represent the same concept, and therefore, they are used interchangeably.

## 4.1. Conformance

A vendor cannot claim conformance to this or any other WfMC specification unless specifically authorised to make that claim by the WfMC. WfMC grants this permission only upon the verification of the particular vendor's implementation of the published specification, according to applicable test procedures defined by WfMC.

Conformance for process definition import / export is essentially based upon conformance to the XPDL grammar as defined by the XML schema (xsd). However, there is a mandatory minimum set of objects, as specified within this document, which must be supported within XPDL. But, given the wide variation of capabilities in modelling tools, it is reasonable to assume that an individual tool might conform to this specification but not be able to swap complete definitions with all other conforming products. A product that claims conformance must generate valid, syntactically correct XPDL, and must be able to read valid XPDL files. A valid, syntactically correct XPDL file must conform and validate against the XPDL schema.

In section 7.2.3 we discuss several aspects of conformance in detail, including the notion of BPMN Model Portability conformance classes.

## 4.2. XPDL Version Compatibility

XPDL version 2.2 is compatible with prior versions, with minor exceptions. The XPDL schema version 2.2 has a different namespace, and tools wishing to be compatible with XPDL version 1.0 need to understand XPDL 1.0 and XPDL 2.2 namespaces.

The following XPDL version 1.0 elements were deprecated in version 2.0:

- Automatic element. Replaced by StartMode and FinishMode attributes of Activity.
- BlockId attribute of BlockActivity element. Replaced by ActivitySetId.
- DeadlineCondition element. Replaced with DeadlineDuration.
- Index attribute in FormalParameter element. Because FormalParameters must match the order in the declaration, and so there is no need for Index.
- Manual element. Replaced by StartMode and FinishMode attributes of Activity.
- Tool Element deprecated.
- Xpression element in Condition element. Replaced by Expression.
- The order in a WorkflowProcess changed from DataFields, Participants, and Applications to be Participants, Applications, and DataFields. This makes the order in Process and package consistent.

Subsequent deprecations (in 2.1 and 2.2) have been done by annotating the XML schema.

## 4.3. References

The following documents are associated with this document and should be used as a reference.

General background information:

WfMC, Terminology & Glossary (WfMC-TC-1011)

WfMC, Reference Model (WfMC-TC-1003)

WfMC, API specifications, which include process definition manipulation APIs:

WfMC, Client Application API Specifications (WAPI) (WfMC-TC-1009)

WfMC, Process Definition Interchange – Process Model (WfMC-TC-1016-P)

BPMI, Business Process Modeling Notation (BPMN), version 1.0 – May 3, 2004

OMG, Business Process Modeling Notation (BPMN) Final Adopted Specification dtc/06-02-01 Feb 2006

OMG, BPMN 1-1 Draft with mark-up DTC-2007-06-02.pdf – June 2, 2007

Object Management Group (OMG), Business Process Model and Notation (BPMN Version 2.0, OMG report: dtc/2010-06-05, OMG, (2010)

OMG Document Number: formal/2011-01-03, <http://www.omg.org/spec/BPMN/2.0>

Process interoperability, used to support process invocation on a remote workflow service:

Workflow Interoperability - Abstract Specifications (WfMC-TC-1012)

Interoperability - Internet E-mail MIME Binding (WfMC-TC-1018)

Accompanying documents:

The Resource Model (Organizational Model: WfMC TC-1016-O)

BPMN icons and tables used in this document are copyright © by the Business Process Management Initiative [BPMI.org], May 3, 2004. All Rights Reserved by BPMI.org.

New BPMN icons, tables and diagrams incorporated from BPMN 1.1 and BPMN 2.0 are copyright © by the Object Management Group (OMG), 2010. All Rights Reserved by OMG.

## 5. Overview of Process Definition Interchange

An XPD L package corresponds to a collection of Process and Collaboration Diagrams in BPMN and consists of a set of Process Definitions, package-level declarations and other package level constructs such as artifacts and message flow between processes. The WfMC defines a process as:

*The representation of a business process in a form that supports automated manipulation, such as modeling, or enactment by a workflow[or business] management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc. (WfMC Glossary - WfMC-TC-1011)*

The process definition provides an environment for a rich description of a process that can be used for the following,

- Act as a template for the creation and control of instances of that process during process enactment.
- For simulation and forecasting.
- As a basis to monitor and analyse enacted processes.
- For documentation, visualization, and knowledge management.

The process definition may contain references to subflows, separately defined, which make up part of the overall process definition.

An initial process definition will contain at least the minimal set of objects and attributes necessary to initiate and support process execution. Some of these objects and attributes will be inherited by each created instance of the process.

The WfMC Glossary also contains descriptions of, and common terminology for, the basic concepts embodied within a process definition such as activities, transitions, relevant data and participants, etc.

### 5.1. Approaches to Process Definition Interchange

This specification uses XML as the mechanism for process definition interchange. XPD L forms a common interchange standard that enables products to continue to support arbitrary internal representations of process definitions with an import/export function to map to/from the standard at the product boundary.

A variety of different mechanisms may be used to transfer process definition data between systems according to the characteristics of the various business scenarios. In all cases the process definition must be expressed in a consistent

form, which is derived from the common set of objects, relationships and attributes expressing its underlying concepts. The principles of process definition interchange are illustrated in Figure 5.: The Concept of the Process Definition Interchange.

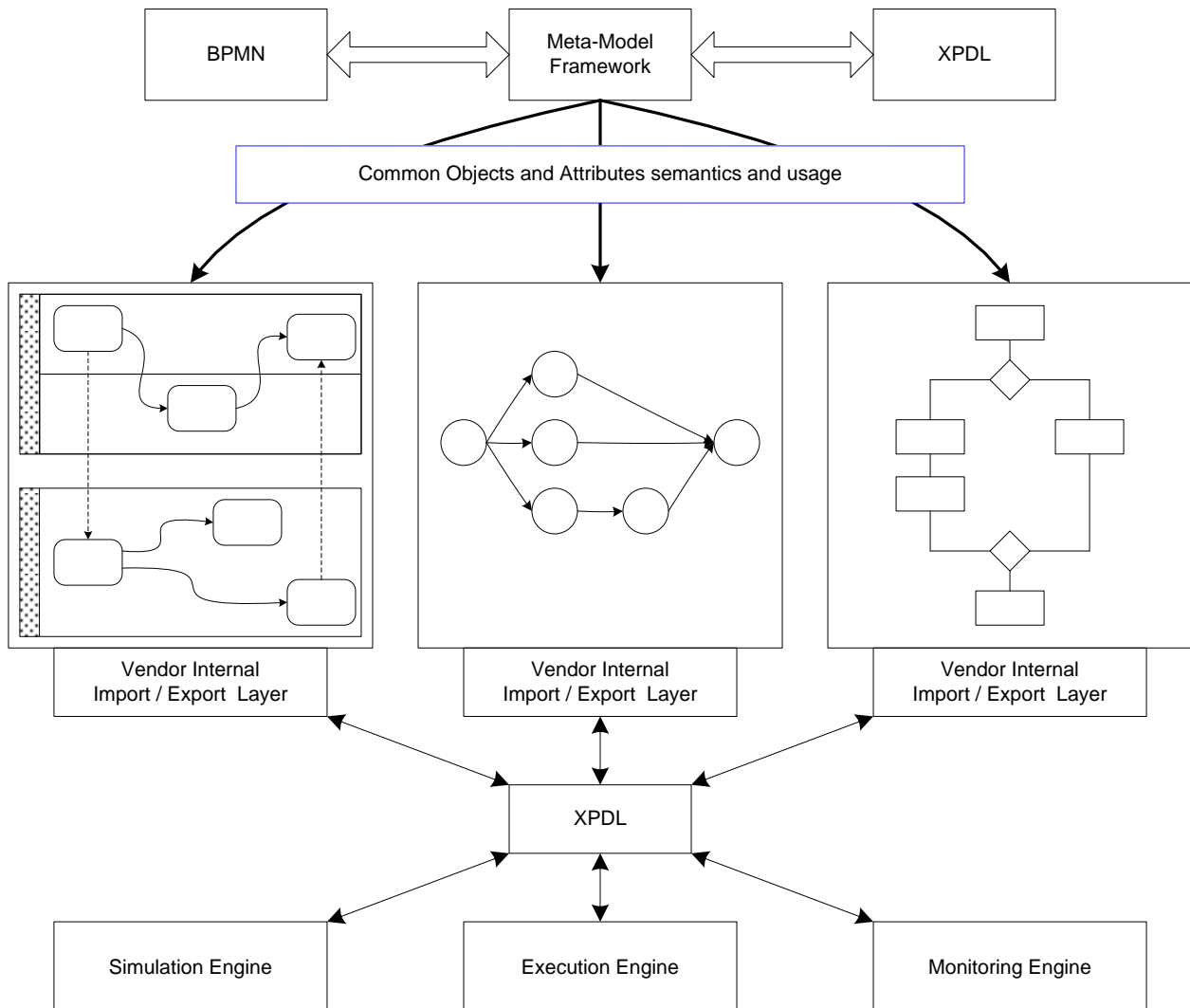


Figure 5.: The Concept of the Process Definition Interchange

## 6. Meta-Model

The Meta-Model describes the top-level entities contained within a Process Definition, their relationships and attributes (including some which may be defined for simulation or monitoring purposes rather than for enactment). It also defines various conventions for grouping process definitions into related process models and the use of common definition data across a number of different process definitions or models.

### 6.1. Processes and Packages

The process model includes various entities whose scope may be wider than a single process definition. In particular the definitions of participants, applications and relevant data may be referenced from a number of process definitions. The meta-model assumes the use of a common process definition repository to hold the various entity types comprising the process definition. Within the repository itself and to support the efficient transfer of process definition data to/from the repository, the concept of a package is introduced, which acts as a container for the grouping of common data entities from a number of different process definitions, to avoid redefinition within each individual process definition.

The package provides a container to hold a number of common attributes from the process definition entity (author, version, status, etc.). Each process definition contained within the package will automatically inherit any common attributes from the package, unless they are separately re-specified locally within the process definition.

An XPDL Package corresponds to a BPMN Business Process Diagram(BPD) in BPMN 1.x. In BPMN 2.0 the concept of a BPD has been removed and several Diagram types have been introduced. XPDL focuses on Process Diagrams and Collaboration Diagrams, which are diagrams in which multiple processes exchange messages. An XPDL package can contain multiple processes and message interchange between any subsets of them.

At the level below Package, there are four new elements which are discussed in later sections of this document:

1. **Pools** (and their lanes) are associated with processes and are used in layout and to define participants (see section 7.4.1) at the Pool/Process Level and performers for the sequence flow elements contained within Lanes.
2. **Message flows** are used to represent communication between processes, based on Web Services Description Language (WSDL) protocols.
3. **Associations** and **Artifacts** are used to document the process definitions. Associations and the Artifacts they connect to provide additional information for the reader of a BPMN Diagram, but do not directly affect the execution of the Process.

BPMN 2.0 adds two new elements at this level: GlobalTask and DataStore (7.1.10.3). Currently using GlobalActivity in XPDL for it. See GlobalActivityReference.

Within a package, the scope of the definitions of some entities is global and these entities can be referenced from all process definitions (and associated activities and transitions) contained within the package. Those entities are:

- participant specification (not a Pool Participant: see section 6.4.1)
- application declaration, and
- DataField

The package reference entity, when used in the package or its contained objects, provides a reference to a top-level entity in the referenced external package:

- Process ids for SubFlow reference
- participant specifications
- application declarations
- type declarations

Conventions on name and identifier management across different packages within the same repository address space to achieve any necessary global uniqueness are for user/vendor definition. The assumed convention during process enactment is that name reference searches follow the sequence:

- Process ids - firstly within the same model (including any references to process definitions for remote execution on a different service), then within any externally referenced model
- Applications / participants - firstly within the same model, then within any externally referenced model

Relevant data naming must be unique within a package; where such data is passed between processes as parameters the convention in this version of specification is that copy semantics will be used. Responsibility rests with process designers / administrators to ensure consistent name / data type usage within process definitions / models to support subflow operations (including any required remote process interoperability).

## 6.2. Package Meta-Model

- Multiple process definitions are bound together in a model definition. The Package acts as a container for grouping together a number of individual process definitions and associated entity data, which is applicable to all the contained process definitions (and hence requires definition only once).

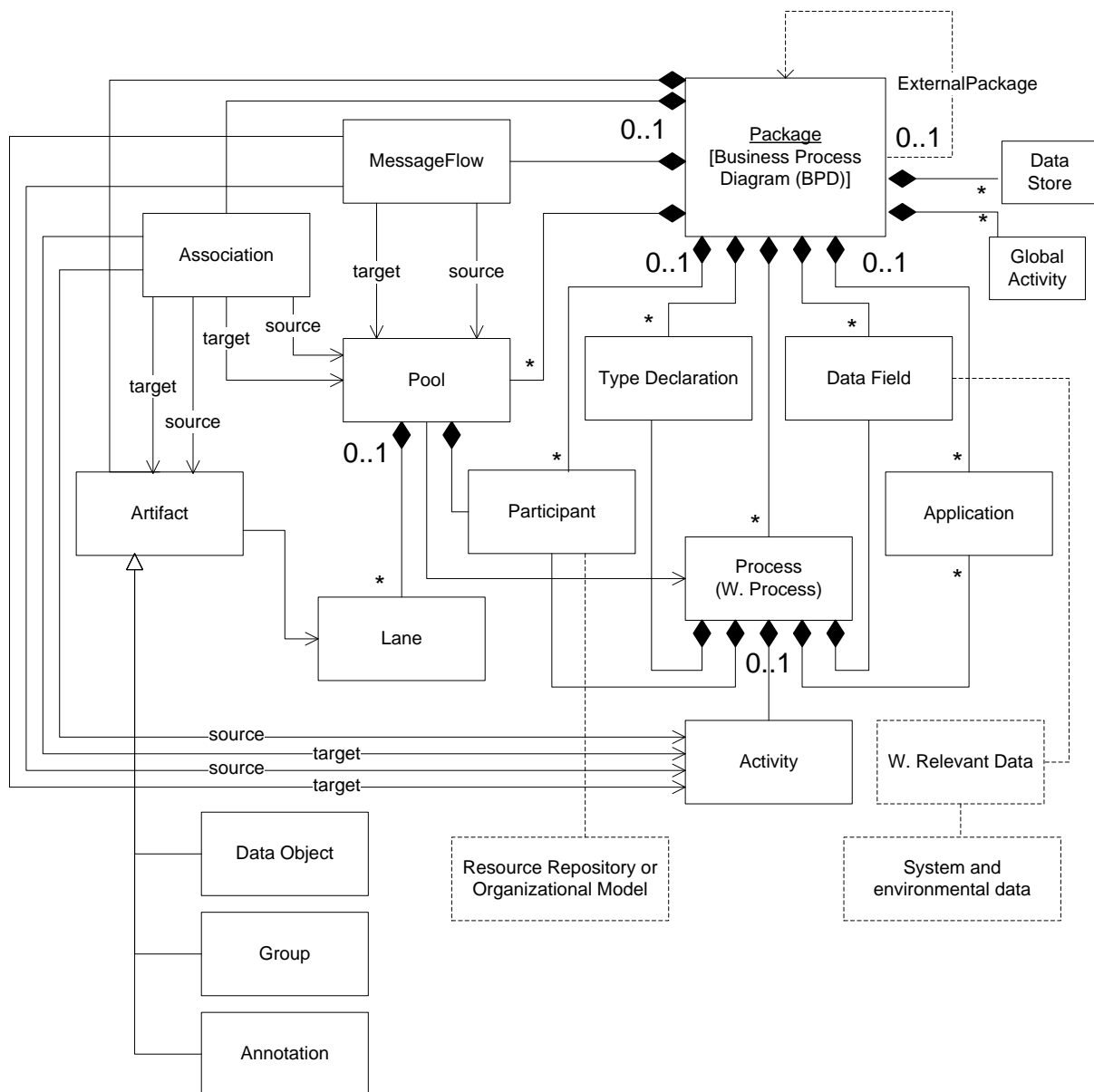


Figure 6.: Package Definition Meta Model

The meta-model for the Package identifies the entities and attributes for the exchange, or storage, of process models. It defines various rules of inheritance to associate an individual process definition with entity definitions for participant specification, application declaration and relevant data field, which may be defined at the package level rather than at the level of individual process definitions.

The Package Definition allows the specification of a number of common process definition attributes, which will then apply to all individual process definitions contained within the package. Such attributes may then be omitted from the individual process definitions. (If they are re-specified at the level of an individual process definition this local attribute

value takes precedence over the global value defined at the package level.)

Note that the Data Object on the Package level is a BPMN1.X element. The BPMN2.0 Data Object is a flow object on the Process Level.

### 6.2.1. Process Repository

The process definition import/export interface is assumed to operate to/from a definition repository of some form associated with the process or workflow management system. The import/export interface is realized by the transfer of files containing XPDL into or out of such a repository. This interface specification allows the import or export of process definition data at the level of individual process definitions and packages.

The internal interface between the repository and control functions is specific to individual vendor products and does not form part of this standard. It is assumed that separation is provided (for example by version control) between repository usage as a static repository (for persistent, ongoing storage of process definition data) and any dynamic usage (for managing changes to the process execution of extant process instances).

The local storage structure of the process definition repository is not part of the WfMC standard. The use of a package is defined only as an aid to simplify the import/export of reusable data structures. Where a simple process repository structure is used, operating at a single level of process definition, shared information within an imported package may be replicated into each of the individual process definitions at the import interface (and similarly repacked, if required, for process definition export).

#### 6.2.1.1. Redefinition and Scope

The possibility of redefining attributes and meta-model entities and referencing external packages introduces the principles of scope and hierarchy into the XPDL (and process repository) structures.

(i) Relevant data field

Process relevant data field has a scope that is defined by the directly surrounding meta-model entity and is not nested. The visibility of its identifier is also defined by that entity.

(ii) Attributes

Attributes including extended attributes have a scope that is defined by the directly surrounding meta-model entity and are nested, i.e. may be redefined at a lower level. Example: The name attribute is redefined in each entity definition. The visibility of extended attribute identifiers is within the particular entity and all sub-entities unless the identifier is redefined in a sub-entity.

(iii) Participants and applications

Participants and applications have a scope and visibility equivalent to extended attributes. All referenced relevant data field and extended attributes have to be defined in the scope where they are used, at least in the same package.

For a referenced external package entity that needs itself reference to entities and their identifiers defined in its external package clause the mechanism is started with the root in that package. That guarantees that no conflict takes place if the invoking process has an entity with the same id, which the definer of the referenced package cannot be aware of.

The described mechanism of external package provides high flexibility for designers and administrators. One can separate organization descriptions (participant entities) and process definitions in separate models, one can add a new release of a process description or add a new process definition sharing the rest of the definition of previously defined and exchanged models without resubmitting the whole context etc.





## 6.4. Entities Overview

The meta-model identifies the basic set of entities used in the exchange of process definitions. The top-level entities are as follows:

### 6.4.1. Swimlanes

Swimlanes are used to facilitate the graphical layout of a collection of processes and the activities they contain. They may designate participant (see section 7.4.1) information at the process level and performer information at the activity level. The Swimlane structure is depicted by a collection of non-overlapping rectangles called Pools. Each Pool may be further subdivided into a number of Lanes.

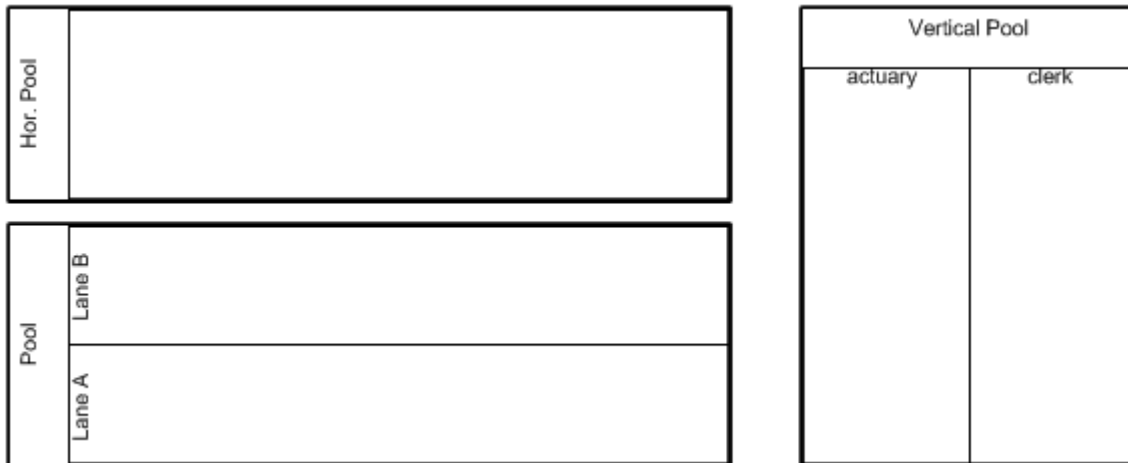


Figure 6.: Swimlanes

BPMN2.0 supports SwimLanes as described for Collaboration Diagrams. BPMN2.0 also introduces the concept of LaneSet as a subelement of Process. We have in XPDL2.2 preserved the original BPMN concept that a Process is always in a Pool and therefore retain the relationship between Pool and Lanes. In a single BPMN2.0 Process Diagram (no collaboration) the Pool will not be visible.

#### 6.4.1.1. Pool

A Pool acts as the container for flow objects (activities) and Sequence Flow (transitions) between them. The Sequence Flow may cross the boundaries between Lanes of a Pool, but cannot cross the boundaries of a Pool. The interaction between Pools is shown through Message Flow.

Another aspect of Pools is whether or not there is any activity detailed within the Pool. Thus, a given Pool may be shown as a “White Box,” with all details exposed, or as a “Black Box,” with all details hidden. No Sequence Flow is associated with a “Black Box” Pool, but Message Flow can attach to its boundaries.

A Pool with flow objects and Sequence Flow contains a Process. An attribute of the Pool identifies the contained Process. (This attribute is null if the Pool is a “Black Box”). The flow objects and Sequence Flow are in that Process, See Process Definition in sections 6.4.2 and 7.5.

Each page of the diagram is regarded as an invisible ‘background pool’. Activities and sequence flow not within an explicit Pool are treated as contained by the background pool. **Some implementations may not support multiple background pools.**

#### 6.4.1.2. Lane

Lanes are used to subdivide a Pool. All the activities within a Lane may inherit one or more properties from the Lane. A typical use of this is to give the Lanes ‘role names’ and have the Activities inherit these role names as ‘Participant assignment/Performer expressions’.

### 6.4.2. Process Definition

The Process Definition entity provides contextual information that applies to other entities within the process. It is a container for the process itself and provides information associated with administration (creation date, author, etc.) or to be used during process execution (initiation parameters to be used, execution priority, time limits to be checked, person to be notified, simulation information, etc.). Note that a BPMN1.x Business Process Diagram (BPD) contains multiple processes. BPMN2.0 supports multiple process definitions.

### 6.4.3. Process Activity

An internal process consists of one or more activities, each comprising a logical, self-contained unit of work. In addition, special activities, referred to as routing activities or Gateways, are used to implement decisions that affect the Sequence Flow path through the process. Special activities, referred to as Events, affect when activities happen and what routes are taken. Events placed on the boundary of an activity can represent alternate terminations of that activity or the instantiation of concurrent subprocesses in the context of the activity.

A typical activity represents work, which will be performed by a combination of resource (specified by performer expressions) and/or computer applications (specified by application assignment). Other optional information may be associated with the activity such as information on whether it is to be started / finished automatically by the process or workflow management system or its priority relative to other activities where contention for resource or system services occurs. Usage of specific relevant data field items by the activity may also be specified. The scope of an activity is local to a specific process definition (although see the description of a subflow activity below).

Note that many of the details about activities are needed only in an execution environment. BPMN diagrams can be drawn and interchanged between design tools without including these details.

In addition to Applications, an activity may be implemented as one of a number of built-in BPMN **tasks**. Their attribute details are discussed in section 7.6.5.3. BPMN diagrams do not require the use of Applications.

An activity may be a subflow - in this case it is a container for the execution of a (separately specified) process definition, which may be executed locally within the same service, or (possibly using the process interoperability interface) on a remote service. The process definition identified within the subflow contains its own definition of activities, internal transitions, resource, and application assignments (although these may be inherited from a common source). In- and out-parameters permit the exchange of any necessary relevant data field between calling and called process (and, where necessary, on return). Such definitions are equivalent to BPMN **Reusable or Callable subprocesses**.

An activity may be a block activity that executes an activity set, or map of activities and transitions. Activities and transitions within an activity set share the name space of the containing process. An activity set is equivalent to a BPMN **embedded subprocess**. BPMN2.0 extends this notion to include **event subprocesses** which can be instantiated by the occurrence of an event either in parallel with a running process or as an interrupt.

An activity may be a route activity, which performs no work processing (and therefore has no associated resource or applications), but simply supports routing decisions among the incoming transitions and/or among the outgoing transitions. BPMN **gateways** are represented by Route activities.

Finally, an activity may represent a BPMN **event**. An event is something that “happens” during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). There are three types of Events, based on when they affect the flow: Start, Intermediate, and End. Their attribute details are discussed in section 7.6.4.

BPMN provides a specific graphical representation for the different activities:

In BPMN 1.x:



In BPMN 2.0:



Figure 6.: Activities

There are additional markers and graphical distinctions for subtypes of each of these representations.

#### 6.4.4. Transition Information

Activities are related to one another via sequence flows (transitions). Each individual transition has three elementary properties, the from-activity, the to-activity and the condition under which the transition is made. Transition from one activity to another may be conditional (involving expressions which are evaluated to permit or inhibit the transition) or unconditional. The transitions within a process may result in the sequential or parallel operation of individual activities within the process. The information related to associated split or join conditions is defined within the appropriate activity, split as a form of “post activity” processing in the from-activity, join as a form of “pre-activity” processing in the to- activity. This approach allows the control processing associated with process instance thread splitting and synchronization to be managed as part of the associated activity, and retains transitions as simple route assignment functions. The scope of a particular transition is local to the process definition, which contains it and the associated activities.

More complex transitions, which cannot be expressed using the simple elementary transition and the split and join functions associated with the from- and to- activities, are formed using route activities, which can be specified as intermediate steps between real activities allowing additional combinations of split and/or join operations. Using the basic transition entity plus route activities, routing structures of arbitrary complexity can be specified. Since several different approaches to transition control exist within the industry, several conformance classes are specified within XPDL. These are described later in the document.

In graphical terms, a transition is a connection between two nodes. BPMN provides three different types of connections, referred to as Sequence Flow, Message Flow and Association. Sequence flow corresponds to Transition as described above. Message Flow and Association are described shortly. BPMN2.0 added DataAssociation which uses the same graphic as Association. BPMN provides a specific graphical representation for the different connections:

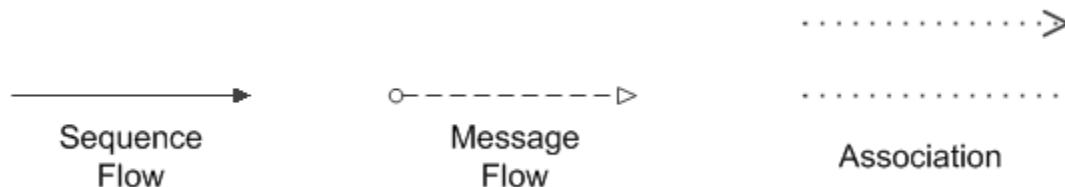


Figure 6.: BPMN Connections

#### 6.4.5. Participant Declaration

This provides descriptions of resources that can act as performers of the various activities in the process definition. The particular resources, which can be assigned to perform a specific activity, are specified as an attribute of the activity, Performers, which links the activity to the set of resources which may be allocated to it. The participant declaration does not necessarily refer to a human or a single person, but may also identify a set of people of appropriate skill or responsibility, or machine automata resource rather than human. The meta-model includes some simple types of resource that may be defined within the participant declaration.

#### 6.4.6. Application Declaration

This provides descriptions of the IT applications or interfaces which may be invoked by the service to support, or wholly automate, the processing associated with each activity, and identified within the activity by an application

assignment attribute (or attributes). Such applications may be generic industry tools, specific departmental or enterprise services, or localized procedures implemented within the framework of the process or workflow management system. The application definition reflects the interface between the engine and the application or interface, including any parameters to be passed. Note that BPMN uses built-in Tasks which make use of Web Services to support or automate the processing, rather than IT applications.

#### 6.4.7. Artifact

To satisfy additional modeling concepts that are not part of the basic set of flow elements (activities, sequence and message flow), BPMN provides the concept of Artifacts that can be linked to the existing Flow Objects through Associations (see below). Thus, Artifacts do not affect the basic Sequence or Message Flow, nor do they affect mappings to execution languages.

At this point, BPMN provides three standard Artifacts: A Data Object (but see below), a Group, and an Annotation. Additional standard Artifacts may be added to the BPMN specification in later versions. A modeler or modeling tool may extend a BPD (Business Process Diagram) and add new types of Artifacts to a Diagram. Any new Artifact must follow the specified Sequence Flow and Message Flow connection rules.

BPMN provides a specific graphical representation for the different artifacts:



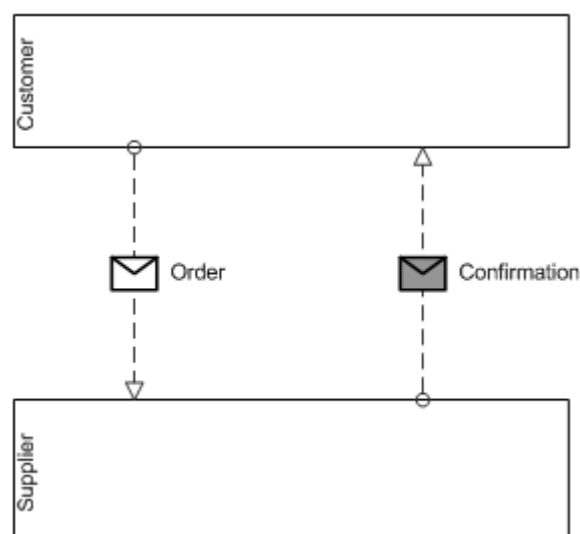
Figure 6.: BPMN Artifacts

Note that in BPMN2.0 DataObject is no longer an artifact but instead a FlowElement. It is used to model dataflow within a process. DataObjects and related elements are discussed in a separate section [7.1.10].

#### 6.4.8. Message Flow

A Message Flow is used to show the flow of messages between two participants/processes (see section 7.4.1) that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two participants/processes (e.g., business entities or business roles). All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

BPMN2.0 introduces explicit representation of the Message on a Message flow.



### **6.4.9. Association**

An Association is used to associate information and Artifacts with Flow Objects. Text and graphical non-Flow Objects can be associated with the Flow Objects and Flow. An Association is also used to show the activities used to compensate for an activity.

An Association does not have a specific mapping to an execution language element. These objects and the Artifacts they connect to provide additional information for the reader of the BPMN Diagram, but do not directly affect the execution of the Process.

The graphics for Association are used in BPMN2.0 also for DataAssociation, which connects DataObject and related elements to model dataflow in a process.

### **6.4.10. Relevant Data Field**

This defines the data that is created and used within each process instance during process execution. The data is made available to activities or applications executed during the process and may be used to pass persistent information or intermediate results between activities and/or for evaluation in conditional expressions such as in transitions or performers (see 6.4.5). Relevant data field is of a particular type. XPDL includes definition of various basic and complex data types, (including date, string, etc.) Activities, invoked applications and/or transition conditions may refer to process relevant data field. The BPMN 2.0 Data Object is an analogous concept see section 7.1.10 .0 for details.

### **6.4.11. Data Types and Expressions**

The meta-model (and associated XPDL) assumes a number of standard data types (string, reference, integer, float, date/time, etc.); such data types are relevant to data fields, system or environmental data or participant data. Expressions may be formed using such data types to support conditional evaluations and assignment of new values to data fields. Data types may be extended using an XML schema or a reference to data defined in an external source.

### **6.4.12. System and Environmental Data**

This is data which is maintained by the process or workflow management system or the local system environment, but which may be accessed by activities or used by the process or workflow management system in the evaluation of conditional expressions and assignments in the same way as relevant data fields.

### **6.4.13. Resource Repository**

The resource repository accounts for the fact that participants can be humans, programs, or machines. In more sophisticated scenarios the participant declaration may refer to a resource repository, which may be an Organizational Model in the case of human participants. Note that this specification does not define or require a resource repository.

### **6.4.14. Vendor or User specific Extensions**

Although the meta-model and associated XPDL contain most of the constructs which are likely to be required in the exchange of process definitions, there may be circumstances under which additional information (user or vendor specific) will need to be included within a process definition. Users and vendors are encouraged to work as far as possible within the standard entity / attribute sets; however, when extensions are needed the XPDL schema provides a standard way to extend it with vendor or user specific extensions.

#### ***6.4.14.1. Extended Elements and Attributes***

The primary method to support such extensions is by the use of extended attributes. Extended attributes are those defined by the user or vendor, where necessary, to express any additional entity characteristics. XPDL schema supports namespace-qualified extensions to all the XPDL elements. The XPDL elements can be extended by adding new child elements, and new attributes.

#### ***6.4.14.2. Extended parameter mapping***

No specific details of the scheme for encoding and passing parameter data are defined within this specification. Where parameters are passed on remote subflow invocation using the workflow Interoperability Specification (interface four),

specifications are provided for the mapping of such parameters (for example into Wf-XML exchanges) using the operations within the concrete syntax specification for interoperability. Any local scheme for parameter mapping and encoding is vendor defined on a product-by-product basis and lies outside the scope of this specification.

## 7. XML Process Definition Language

### 7.1. Elements Common for Multiple Entities

#### 7.1.1. Id Attribute of Many Objects

All IDs for objects should be unique within a package (XPDL file). For the package external ref feature (see section 7.2.5) the id must be qualified by a 'namespace' attributed to the external package so id-clash is avoided.

The use of Id and IdRef allows the schema validation to detect violations, either duplicate Id's or IdRefs to non-existent elements.

##### 7.1.1.1. Graphic Information

Graphic information is optional and tool dependent. Graphical portability requires this information. The graphic information (NodeGraphicsInfo and ConnectorGraphicsInfo) may appear multiple times on each XPDL element, depending on the number of tools that have added the graphical information to the XPDL file. Each tool, identified by ToolId, can add its own graphical information. Therefore, each tool can display and represent the same XPDL in totally different ways, because each tool uses its own graphical information, but retains the graphical information from other tools.

We recommend that where graphical portability between different products is important, the use of ToolId should be limited. Recommended use cases are as follows:

- ToolId is never used or exactly the same ToolId appears everywhere.
  - Only one GraphicsInfo appears for each element.
- Multiple ToolId's occur or a mix of at least one ToolId and cases where no ToolId is given (null ToolId case).
  - When a tool loads the file, it asks which ToolId to prefer.
    - If an element has GraphicsInfo with that ToolId, that Graphics Info is used.
    - Otherwise, choose the GraphicsInfo with no ToolId if present.
      - Otherwise, ignore GraphicsInfo.

##### 7.1.1.2. Pages

A complex business process may best be represented on multiple pages for viewing and printing purposes. We have added explicit support for this purpose. All graphical elements can refer to the page structure by using the PageId attribute. Pages allow the user to control on which Page each Process and Sub-Process appears.

Schema in XSD file.

	Description
Height	Height of the page in pixels.
Id	Id of the page. Referred to in all Graphics to designate the page.
Name	Name of the page.
Width	Width of the page in pixels..

Table : Pages

##### 7.1.1.3. Coordinate Information

Where coordinate information is provided, the following interpretation applies.

All co-ordinates have origin of top-left, relative to the page the object is on. No negative coordinates. Coordinates for any type of Node (Pool, Lane, Activity and so forth) designate the position of the top-left corner of the bounding box for the Node, relative to the top left corner of the Page.

Scale: Co-ordinate units are in pixels. However it would be nice to give other applications a hint as to the scale of a

'pixel' when the XPD L file was saved (i.e. The XPD L writer specifies co-ordinates and sizes in pixels but can also specify 'pixels to the millimeter' - the reading application can then, if it wishes, take this into account and scale to its pixel size appropriately).

This layout information is provided by the element 'LayoutInfo' and attribute 'PixelsPerMillimeter'. See PackageHeader (section 7.2.1).

#### 7.1.1.4. NodeGraphicsInfo

NodeGraphicsInfo is an optional entity that can be used by a tool to describe graphical information. Each graphical node in XPD L (Activity, Pool, Lane, Artifact) has a list of NodeGraphicsInfo, one for each tool that has saved the corresponding information in the XPD L file. If a tool chooses to use the NodeGraphicsInfo, it should select a tool id, which can be the name of the tool. Therefore multiple tools can work using the same XPD L file and displaying the process with a different presentation layout.

For Portability of diagram graphics, we recommend all elements that allow NodeGraphicsInfo as a child element in fact have that element with attributes PageId, Width, Height and Coordinate information provided. Further, if the element are positioned within a Lane, then the LaneId should be provided as well.

Schema in XSD.

	Description
BorderColor	Color of the border, expressed as a string. Tool specific and depends on ToolId.
Coordinates	X and Y coordinates of node's upper left corner (bounding box). Tool specific and depends on ToolId. Usual implementation based on upper left corner of page being (0, 0) and all coordinates >= 0.
FillColor	Color of the fill, expressed as a string. Tool specific and depends on ToolId.
Height	Height of the node. Tool specific and depends on ToolId.
LaneId	If the node is in a Lane, this is the Id of the Lane.
ToolId	Tool id. This may correspond to the name of the tool generating the XPD L file. Note that multiple NodeGraphicsInfo elements may appear in an element. This allows each tool to have its NodeGraphicsInfo, allowing for different tools using the same XPD L file to represent the element in totally different ways. Each tool reads and writes its own NodeGraphicsInfo based on its ToolId, and it leaves untouched the NodeGraphicsInfo from other tools.
IsVisible	True indicates node should be shown.
Page [Deprecated]	The name of the page on which this node should be displayed. Used in XPD L 2.0 without sufficient guidelines to insure consistency and portability.
PageId	The id of the page on which this node should be displayed. Refers to Pages element described in section 7.1.1.2.
Shape	Shape, expressed as a string: used to override BPMN shapes.
Width	Width of the node. Tool specific and depends on ToolId.

Table : Node Graphics Info

#### 7.1.1.5. ConnectorGraphicsInfo

ConnectorGraphicsInfo is an optional entity that can be used by a tool to describe graphical information for connecting objects (SequenceFlow, MessageFlow, Associations). Each connector in XPD L has a list of ConnectorGraphicsInfo, one for each tool that has saved the corresponding information in the XPD L file. If a tool chooses to use the ConnectorGraphicsInfo, it should select a tool id, which can be the name of the tool. Therefore multiple tools can work using the same XPD L file and displaying the process with a different presentation layout.

All co-ordinates have origin of top-left, relative to the page. The co-ordinates list should include the path end points (clipped to the boundary of the source and destination nodes as applicable). If the co-ordinate path is omitted it is assumed that the path is a single segment between the centers of the source and destination nodes (clipping is applied as



needed to the endpoints).

For Portability of diagram graphics, we recommend all elements that ConnectorGraphicsInfo as a child element in fact have that element with attributes PageId and Coordinate information provided.

Schema in XSD file.

	Description
BorderColor	Color of the border, expressed as a string. Tool specific and depends on ToolId.
Coordinates	X and Y coordinates of points in the path. Tool specific and depends on ToolId.
FillColor	Color of the border, expressed as a string.
ToolId	Tool id. This may correspond to the name of the tool generating the XPD file. Note that multiple ConnectorGraphicsInfo elements may appear in an element. This allows each tool to have its ConnectorGraphicsInfo, allowing for different tools using the same XPD file to represent the element in totally different ways. Each tool read and writes its own ConnectorGraphicsInfo based on its ToolId, and it leaves untouched the ConnectorGraphicsInfo from other tools.
IsVisible	True indicates connector should be shown.
Page [Deprecated]	The name of the page on which this connector should be displayed. Used in XPD 2.0 without sufficient guidelines to insure consistency and portability.
PageId	The id of the page on which this node should be displayed. Refers to Pages element described in section 7.1.1.2.
Shape	Shape, expressed as a string: used to override BPMN shapes.
Style	LineStyle. Tool specific and depends on ToolId. Should not conflict with BPMN recommended styles for SequenceFlow, MessageFlow and Association.

*Table : Connector Graphics Info*

### 7.1.2. Expression Type

The ExpressionType specifies the scripting language and particular script expression for a given XPD expression. The script expression itself is contained within the body of the containing element. All three of the attributes need only be specified if they wish to override that specified in the Package Header. For compatibility with BPMN2.0 the default is XPATH.

ExpressionType is referred to in many elements that have expressions.

Schema in XSD file.

	Description
Type	Identifies the scripting language used in expressions. For consistency across implementations, when specifying a standard scripting language, it is recommended that the Type be selected from the following strings: text/javascript, text/vbscript, text/tcl, text/ecmascript, text/xml, application/xslt+xml.
Version	This is the version of the scripting language.
Grammar	This is a reference to a document that specifies the grammar of the language. It could be, for example, an XML schema, a DTD, or a BNF.

*Table : ExpressionType*

### 7.1.3. Extended Attributes

Extended Attributes can be used in all entities. They allow vendors to extend the functionality of this specification to meet individual product needs. A vendor can use extended attributes in two ways:

- a- The anonymous extended attribute provides a name and a value for the extension without the need to qualify the extension with a namespace.
- b- Namespace qualified extensions can be used to extend XPDL elements by adding child elements or by adding attributes. Namespace qualified extensions can be validated against a vendor provided schema (see section 7.2.1.1 Vendor extensions).

#### 7.1.3.1. Anonymous Extended Attribute

The anonymous extended attributes use the ExtendedAttribute element to add an extension with a name and a value.

Schema in XSD file.

	Description
Name	Used to identify the Extended Attribute.
Value	Value of the extension.

Table : Extended Attributes

#### 7.1.3.2. Namespace Qualified Extensions

In order for a tool to add namespace-qualified extensions, it first needs to extend the XPDL schema to add the extensions in the places in which the XPDL schema allows the tool to add extensions. The new generated schema should contain in addition to the XPDL namespace the tool namespace. The resulting schema can be used in addition to the XPDL schema to validate XPDL 2.1 files. The extension places are marked in the XPDL schema by [namespace="##other" processContents="lax"].

In addition, the tool should include the namespace, and it should add the VendorExtension section in XPDL (see 7.2.1.1 Vendor extensions), which contains a URI reference to the extended schema.

A child element can be added to a XPDL element, by adding a child element to the ExtendedAttributes under the XPDL element. An attribute to a XPDL element can also be added by just qualifying it by the vendor namespace.

An example of how a vendor can create a schema with XPDL extensions is provided in section .

#### 7.1.4. Formal Parameters

Formal parameters can be used as attributes in processes and activities. They are passed during invocation and return of control (e.g. of an invoked application). These are the invocation parameters.

Schema in XSD file.

	Description
DataType	Data type of the formal parameter. See section 7.13.
Description	Textual description of the formal parameter.
Length	The length of the data. Used only for strings, to declare maximum length.
Id	Identifier for the parameter.
InitialValue	Pre-assignment of data for runtime.
Name	Name for the parameter.
IsArray	Indicates if the parameter is an array or a single value parameter.
Index	Index of the parameter (Deprecated)
Mode	<div>IN                      Input Parameters</div> <div>OUT                    Output Parameters</div> <div>INOUT                  Parameters used as input and output</div>

	Description
ReadOnly	The datafield or formal parameter is described as readOnly or as a constant and its value cannot be changed.
Required	Defines whether the parameter must be supplied, default to false.

Table : Formal Parameters

#### 7.1.4.1. Parameter passing semantics

The parameter passing semantics is defined as:

- (a) Any read-only formal parameters (IN) are initialised by the value of the corresponding actual parameter in the call (an expression). This is pass-by-value semantics.
- (b) Any read/write formal parameters (INOUT) are initialised by the value of the corresponding actual (passed) parameter, which must be the identifier of a relevant data field entity. On completion of the process, the value of the formal out parameter is copied back to the original actual parameter (which must be the identifier of a relevant data field entity). This is copy-restore semantics.
- (c) Any write-only formal parameters (OUT) are initialised to zero (strings will be set to the empty string, complex data will have each element set to zero). On completion of the process, the value of the formal out parameter is copied back to the original actual parameter (which must be the identifier of a relevant data field entity). This is zero-restore semantics.

#### 7.1.4.2. Concurrency semantics

Copying and restoring of parameters are treated as atomic operations; to avoid access conflicts from concurrent operations on relevant data field within the process instance these operations are serialized. Between copy and restore of (c) no locking is assumed and the returned parameter value will overwrite the local value (of the particular relevant data field item) at the time of the return call.

### 7.1.5. External Reference

ExternalReference is a reference to an external definition of an entity. It can be used in DataTypes, Participant, and Application.

Schema in XSD file.

	Description
Location	It specifies the URI of the document that defines the type.
Namespace	It allows specification of the scope in which the entity is defined.
Xref	It specifies the identity of the entity within the external document.

Table : External Reference

Example 1: A FormalParameter that is defined by an XML schema:

```
<FormalParameter Id="PO">
  <DataType>
    <ExternalReference location="http://abc.com/schemas/po.xsd"/>
  </DataType>
  <Description>PO specification for abc.com</Description>
</FormalParameter>
```

Example 2: A DataField defined by a Java class:

```
<DataField Id="PO" Name="PurchaseOrder" IsArray="FALSE">
  <DataType>
    <ExternalReference location="com.abc.purchases.PO"/>
  </DataType>
  <Description>PO specification for abc.com</Description>
</DataField>
```

### 7.1.5.1. Web Services

An activity in a process may invoke a web service. The ExternalReference element may be used as a reference to applications and data types that are defined in Web Service (WSDL) documents.

Example 3: A DataField whose data type is defined in a WSDL document:

```
<DataField Id="abcPO" Name="abcPurchaseOrder" IsArray="False">
  <DataType>
    <ExternalReference xref="PO" location="http://abc.com/services/poService.wsdl" namespace="poService/definitions/types"/>
  </DataType>
</DataField>
```

Example 4: An Application that is defined as an operation in a WSDL document:

```
<Application Id="placeOrder">
  <ExternalReference location="http://abc.com/PO/services/poService.wsdl"
    xref="PlaceOrder" namespace=
      "http://abc.com/services/poService.wsdl/definitions/portType"/>
</Application>
```

## 7.1.6. Actual Parameters

Actual parameters hold the information required to map data held in a process to a formal parameter of a callable activity (sub-process or activity).

The core concept of an actual parameter is one or more sources, a target, and an optional formal expression defining how the data should be copied into the formal parameter.

### BPMN 2.0 compatibility

BPMN 2.0 Activities and Events must check if data elements are *available* before executing. XPDL formal parameters are always initialized and hence available, see section 7.1.4.1 Parameter passing semantics. Data Fields are initialized at the time the process is created, similar to Parameter OUT semantics.

	Description
Assignments	Optional reference to one or more Assignment expression used to map between different type systems used by formal and actual parameters. Each assignment maps a single, atomic piece of data from the source structure to the target.
FormalExpression	An optional expression used to map between different type systems used by formal and actual parameters. Differs from Assignment by defining the entire target structure in one go.
Id	Identifier for the actual parameter. Type is xpdل:IdRef
Source	Reference to an in-scope data item (for example DataField or process FormalParameter)
Target	Reference to the FormalParameter that will be copied to or from by this actual parameter. Type is xpdل:IdRef

### 7.1.6.1. Formal-actual Parameter Mapping

The mapping of actual to formal parameters during invocation is defined by a parameter map list. In the simplest case, the actual parameters are mapped 1:1 to the formal parameters in sequence, i.e. the first actual maps to the first formal, the second actual maps to the second formal etc. Type compatibility is required within the definitions and may be enforced by the run-time system. The effects of violation are locally defined and do not form part of this specification.

If the type systems of formal and actual parameters differ the following mechanisms exist to map between the type systems:

- Provide a formal expression as part of the actual parameter ;
- Provide one or more Assignments, see section 7.1.7 ;
- Provide a DataMapping, see section 7.6.5.4.7 (deprecated in XPDL 2.2).

If the actual parameter contains an expression the direction of mapping is controlled by the Formal Parameter's mode.

- If the targeted formal parameter is mode IN this is an expression defining the value of the formal parameter (or part of it) in the form *target = formal expression*. There MAY be zero to many sources defined in this case, but there is no requirement that these sources are used inside the expression.
- Otherwise, if the targeted formal parameter is mode OUT this expression defines the value of the actual parameter in the form *source = formal expression*.
- FormalExpression MUST NOT be used when the target formal parameter has mode = INOUT meaning that the type of both formal and actual parameter must be the same for this mode. This is commonly used when invoking a sub-process as parent and child processes are likely to share the same type system.

How any buffering and mapping is performed by the run-time system is outside the scope of this document. . Refer to section 7.1.2 for more information on expression types. Refer to the new section on BPMN2.0 Data Objects (7.1.10) for more details specific to BPMN 2.0.

### BPMN 2.0 Compatibility

BPMN 2.0 contains the idea of alternative “input sets” of data for a task. Each defined input set is evaluated in the order they are included in the BPMN InputOutputSpecification element of the task. If all the referenced data inputs are valid the task is invoked using that set of data. However, if not all data is available, then the next input set will be evaluated and so on until none are left. If no input set is valid then the task will wait. BPMN 2.0 states that determining how or how often a waiting task is re-evaluated is out of scope.

XPDL’s formal-actual parameter mapping does not define an equivalent way to define multiple data contracts for the same task in this version.

#### 7.1.7. Assignment

Data fields may be explicitly assigned new values during the execution of a process. This may happen at the start or termination of a process or an activity. Assignments may also be associated with transitions/sequence flow, in which case the assignments are performed when the particular transition/outgoing sequence flow is chosen.

BPMN2.0 introduces DataObjects and DataAssociations as a graphically explicit approach to Assignment. See section 7.1.10.

Schema in XSD file.

	Description
Target	Lvalue expression, normally the Id of a Data Field or Formal Parameter.
Expression	Rvalue expression in the scripting language specified for the package.
AssignTime	Specifies time of evaluation and assignment: Start or End of Process or Activity. Not relevant to transition/sequence flow or to actual parameters.

Table : Assignment

#### 7.1.8. Category

The modeler MAY add one or more Categories to various elements, such as Process, Activity and Transition; which can then be used for purposes such as reporting and analysis. In OLAP-based reports the categories would be members of the category dimension and allow filtering and slice-and-dice queries.

Schema in XSD file.

	Description
Id	Id of the category.
Name	Name of the category.

Table : Category

### 7.1.9. Artifact

BPMN provides modelers with the capability of showing additional information about a Process that is not directly related to the Sequence Flow or Message Flow of the Process.

At this point, BPMN provides three standard Artifacts: A Data Object (but see below), a Group, and an Annotation. Additional standard Artifacts may be added to the BPMN specification in later versions. A modeler or modeling tool may extend a BPD and add new types of Artifacts to a Diagram. Any new Artifact must follow the Sequence Flow and Message Flow connection rules (listed below). Associations can be used to link Artifacts to Flow Objects.

An Artifact is a graphical object that provides supporting information about the Process or elements within the Process. However, it does not directly affect the flow of the Process. Other examples of Artifacts include critical success factors and milestones.

BPMN provides a specific graphical representation for the different artifacts:



Figure 7.: Artifacts

Note that BPMN2.0 makes DataObject a flow element. See section below on DataObject.

#### 7.1.9.1. Sequence Flow Connections for Artifacts

- An Artifact MUST NOT be a target for Sequence Flow.
- An Artifact MUST NOT be a source for Sequence Flow.

#### 7.1.9.2. Message Flow Connections for Artifacts

- An Artifact MUST NOT be a target for Message Flow.
- An Artifact MUST NOT be a source for Message Flow.

#### 7.1.9.3. Schema for Artifacts

All schema fragments are contained in a separate XSD document. Here we describe important attributes of the element and its sub-elements.

	Description
Id	Id of the artifact.
Name	Name of the artifact.
ArtifactType	DataObject   Group   Annotation
DataObject	See section 7.1.9.5.
Group	Name of the Group. Attribute deprecated in 2.1. Replaced by Group Element. See section 7.1.9.6.
NodeGraphicsInfos	See section 7.1.1.
Object	See section 7.1.9.4.
TextAnnotation	Visible textual description.

Table : Artifact

#### 7.1.9.4. Object

Referred to by numerous other graphical elements.

	Description
Name	Name of the Object.
Categories	A list of categories. See section 7.1.8.
Documentation	Textual Documentation.
Id	Id of the object.

Table : Object

#### 7.1.9.5. DataObject in BPMN1.x

In BPMN1.x, a Data Object is considered an Artifact and not a Flow Object. Refer to section 7.1.10 for a discussion of DataObject in BPMN2.0. It is considered an Artifact because it does not have any direct affect on the Sequence Flow or Message Flow of the Process, but it does provide information about what the Process does. That is, how documents, data, and other objects are used and updated during the Process. While the name “Data Object” may imply an electronic document, it can be used to represent many different types of objects, both electronic and physical.

As an Artifact, Data Objects generally will be associated with Flow Objects. An Association will be used to make the connection between the Data Object and the Flow Object. This means that the behavior of the Process can be modeled without Data Objects for modelers who want to reduce clutter. The same Process can be modeled with Data Objects for modelers who want to include more information without changing the basic behavior of the Process.

In some cases, the Data Object will be shown being sent from one activity to another, via a Sequence Flow. Data Objects will also be associated with Message Flow. They are not to be confused with the message itself, but could be thought of as the “payload” or content of some messages.

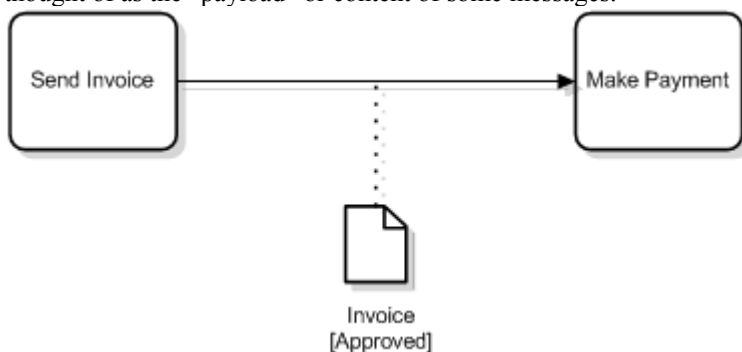


Figure 7.: A Data Object associated with a Sequence Flow

In other cases, the same Data Object will be shown as being an input, then an output of a Process. Directionality added to the Association will show whether the Data Object is an input or an output. Also, the state attribute of the Data Object can change to show the impact of the Process on the Data Object.

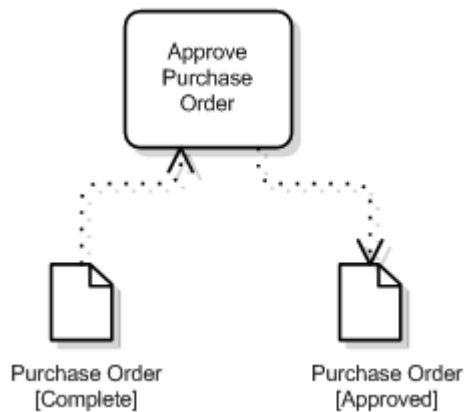


Figure 7.: Data Objects shown as inputs and outputs

	Description
DataFieldRef	A list of data fields.
Id	Id of the data object. Type is xpdId.
Name	Name is an attribute that is a text description of the object.
NodeGraphicsInfos	Needed for BPMN2.0 DataObjects, which are not artifacts and therefore require a sub-element for graphical information. See section 7.1.9.7 for BPMN2.0 DataObject and section 7.1.1 for discussion of NodeGraphicsInfos.
Object	Used to associate categories and documentation with flow object.
State	State is an optional attribute that indicates the impact the Process has had on the Data Object. Multiple Data Objects with the same name MAY share the same state within one Process.

Table : Data Object

#### 7.1.9.6. Group

The Group object is an Artifact that provides a visual mechanism to group elements of a diagram informally. The grouping is tied to the Category supporting element (which is an attribute of all BPMN elements). That is, a Group is a visual depiction of a single Category. The graphical elements within the Group will be assigned the Category of the Group. (Note -- Categories can be highlighted through other mechanisms, such as color, as defined by a modeler or a modeling tool).



Figure 7.: A Group Artifact

As an Artifact, a Group is not an activity or any Flow Object, and, therefore, cannot connect to Sequence Flow or Message Flow. In addition, Groups are not constrained by restrictions of Pools and Lanes. This means that a Group can stretch across the boundaries of a Pool to surround Diagram elements (see Figure below), often to identify activities that exist within a distributed business-to-business transaction.



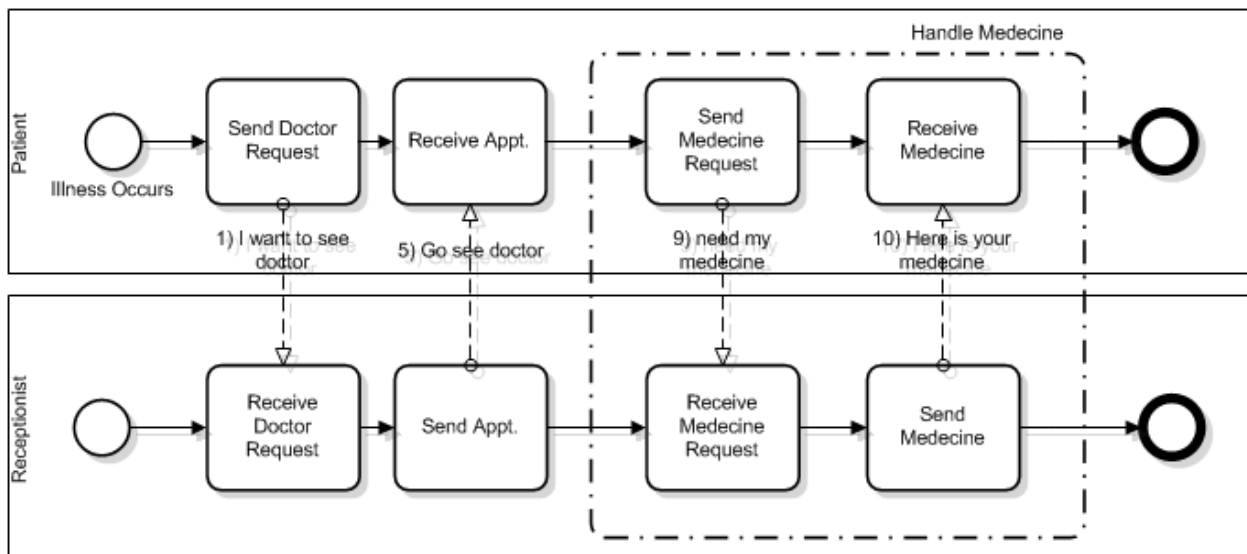


Figure 7.: A Group around activities in different Pools

Groups are often used to highlight certain sections of a Diagram without adding additional constraints for performance--as a Sub-Process would. The highlighted (grouped) section of the Diagram can be separated for reporting and analysis purposes. Groups do not affect the flow of the Process.

	Description
Category	Category specifies the Category that the Group represents (Further details about the definition of a Category can be found in section 7.1.8). The name of the Category provides the label for the Group. The graphical elements within the boundaries of the Group will be assigned the Category.
Id	Id of the Group.
Name	Name is an attribute that is the text description of the Group.
Object	A list of the IDs of all the Objects in the Group.

Table : Group

## 7.1.10. Data Object for BPMN2.0

### 7.1.10.1. Data Modeling

A traditional requirement of Process modeling is to be able to model the data that is created, manipulated, and used during the execution of a Process. This requirement is realized in BPMN through various constructs: Data Objects, ItemDefinition, Properties, Data Inputs, Data Outputs, Messages, Input Sets, Output Sets, and Data Associations. This section shows how the same data modeling requirements are met in XPD and how this maps to the BPMN constructs.

#### BPMN Item-Aware Elements

Several elements in BPMN are used to store or convey data items during process execution. These elements are referenced generally as “item-aware elements.” This is similar to the variable construct common to many languages. The data structure these elements hold is specified using an associated ItemDefinition. An item-aware element may be underspecified, meaning that the structure attribute of its ItemDefinition is optional if the modeler does not wish to define the structure of the associated data. The elements in the specification defined as item-aware elements are: Data Objects, Properties, DataInputs and DataOutputs, DataStore, Messages.

In XPD2.2 FormalParameters of a Process, ActivitySet or Global Task and DataFields are used to define data structure

and may be underspecified as in BPMN. Refer to section 6.4.10: Relevant Data Field. Activities use FormalParameters to declare a data contract analogous to the BPMN xxxTask elements that refer to ItemDefinitions through the itemSubjectRef of their DataInput and DataOutputs. Refer to section 7.1.4: Formal Parameters. Actual parameters of XPDL activities provide the mapping of in and out of relevant data similarly to BPMN dataInputAssociations and dataOutputAssociations. This data mapping includes any necessary transformation between type systems using XPDL expressions, refer to section 7.1.6.1: Formal-Actual parameter mapping. This version of the XPDL specification has no direct equivalent to BPMN Properties. Refer to section 0: Purpose.

### Producing and Consuming Data Objects

Activities and Processes often required data in order to execute. In addition they may produce data during or as a result of execution. Data requirements are captured as Formal Parameters with mode IN. Data that is produced is captured using Formal Parameters with mode OUT. Execution semantics, including initialization, are defined for Formal Parameters in section 7.1.4.1 and they apply the same way to all elements that extend it. Not every Activity type defines inputs and outputs, only Tasks, CallableElements (Global Tasks and Processes) can define their data requirements.

Note that Formal Parameters of Activities are new in XPDL 2.2 and replace Input and Output Sets in order that Activities and Processes are consistent. Note that XPDL provides two methods to map data to this Formal Parameter at execution time. These two methods are defined as Actual Parameters (see section 7.1.6) and DataMapping (see section 7.6.5.4.7).

#### 7.1.10.2. Data Object

The primary visualization of data within the process flow is the DataObject element. Data definition is performed by Formal Parameters to the Process or Data Fields within it but it can be convenient to have several visual representations of the same data to make the drawing of associations to different activities clear and avoid detracting from the process flow.

Data Object elements must be contained within Process or Sub- Process elements. In XPDL this means that the DataObjects element is a child of Process or ActivitySet. Data Object elements are visible in a Process diagram.

	Description
DataFieldRef	Id of a data field that this DataObject is the visual representation of. Note that the schema allows for more than one DataFieldRef but when used as a BPMN 2-style DataObject there MUST be zero or one DataFieldRef.
Id	Id of the data object. Type is xpdId.
Name	(optional) Name is an attribute that is a human readable name of the object.
NodeGraphicsInfos	Needed for BPMN2.0 DataObjects, which are not artifacts and therefore require a sub-element for graphical information. See section 7.1.9.7 for BPMN2.0 DataObject and section 7.1.1 for discussion of NodeGraphicsInfos.
Object	Used to associate categories and documentation with flow object.
State	(optional) Indicates the condition of the Data Object at this point in the Process. Multiple Data Objects with the same name MAY share the same state within one Process.

A Data Object element that references a Data Field marked with IsArray="true" has to be visualized differently, compared to single *instance* data structures.



Single Instance



Collection

### Visual representations of Data Objects

Data Objects can appear multiple times in a Process diagram. Each of these appearances references the same Data Field *instance*. Multiple occurrences of a Data Object in a diagram are allowed to simplify diagram connections.

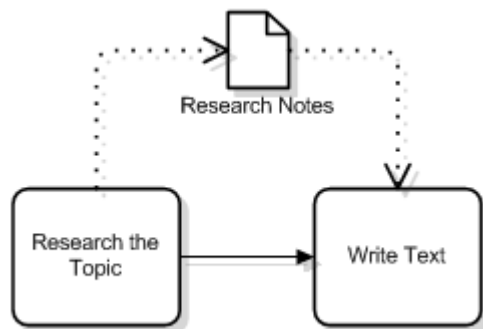


Figure : Data Object produced by one activity and consumer by another

Alternatively, Data Objects may be directly associated with a Sequence Flow connector to represent the same relationships. This is a visual short cut that normalizes two Data Associations – one from an item-aware element (e.g., an Activity) contained by the source of the Sequence Flow, connecting to the Data Object; and the other from the Data Object connecting to an item-aware element contained by the target of the Sequence Flow.

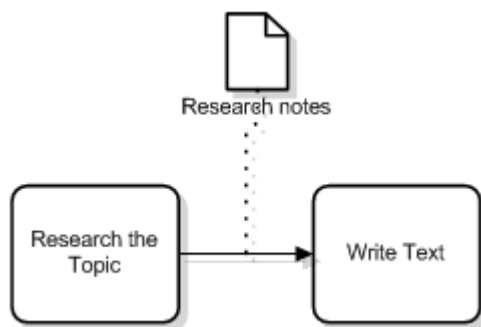


Figure : Alternate visualization of Data Object associated directly with Sequence Flow

#### 7.1.10.3. Data Store

A DataStore provides a mechanism for Activities to retrieve or update stored information that will persist beyond the scope of the Process. In contrast to Data Fields, these allow an Activity to control when data is loaded into the process and therefore solve data-freshness issues. The XPDL DataStores element is a child of Package and contains a list of all DataStore elements for the Package.

	Description
Capacity	Integer.
Id	Id of the DataStore. Type is xpdId.
IsUnlimited	If isUnlimited is set to true, then the capacity of a Data Store is set as unlimited and will override any value of the capacity attribute.
Name	(optional) Name is an attribute that is a human readable name of the object.
Object	Used to associate categories and documentation with flow object.

#### 7.1.10.4. Data Store Reference

The same Data Store can be visualized, through a Data Store Reference, in one or more places in the Process. The Data Store Reference is an ItemAwareElement and can thus be used as the source or target for a Data Association. When data flows into or out of a Data Store Reference, it is effectively flowing into or out of the DataStore that is being referenced. The XPDL DataStoreReferences element is a child of Process and ActivitySet and contains a list of all DataStoreReference elements for the Process or ActivitySet.



	Description
DataStoreRef	IdRef of DataStore
Id	Id of the DataStoreReference element. Type is xpdId.
Name	(optional) Name is an attribute that is a human readable name of the object.
NodeGraphicsInfos	Graphical information about flow object in the diagram
Object	Used to associate categories and documentation with flow object.

State (optional) Indicates the condition of the Data Store at this point in the Process. Multiple Data Stores with the same name MAY share the same state within one Process.

#### 7.1.10.5.Data Input

A Data Input is a visualization of one incoming Formal Parameter of a Process or Activity. There may be multiple Data Inputs associated with a Process or Activity just as there may be more than one Formal Parameter. Data Inputs may appear in a Process diagram to show the inputs to the Process as a whole and are passed along as the inputs of Activities by Data Associations.

- Data Inputs have the same notation as Data Objects, except MUST contain a small, unfilled block arrow
- Data Inputs MUST NOT have incoming Data Associations except if the process they belong to is being called from another, external process. In this case a Data Association MAY be used to visualize where the Data Input comes from.

	Description
FormalParameterRef	IdRef of FormalParameter this DataInput depicts. The Formal Parameter referred MUST have Mode="IN".  <i>Note:</i> Whilst there may be zero or more formal parameter references this is for purely historic reasons to mirror DataFieldRef in DataObjects, which is there to support BPMN 1.2 style DataObjects. Exactly one formal parameter reference is the expected usage.
Id	Id of the DataInput element. Type is xpdId.
Name	(optional) Name is an attribute that is a human readable name of the object.
NodeGraphicsInfos	Graphical information about flow object in the diagram.
Object	Used to associate categories and documentation with flow object.
State	(optional) Indicates the condition of the Data Input at this point in the Process. Multiple Data Inputs with the same name MAY share the same state within one Process.

A Data Input element that references a Formal Parameter marked with IsArray="true" has to be visualized differently, compared to single *instance* data structures.



Single instance    Collection

#### BPMN 2.0 compatibility

In BPMN 2, Data Inputs have an "optional" attribute that defines if a DataInput is valid even if the state is "unavailable". The default value is false. If the value of this attribute is true, then the execution of the Activity will not begin until a value is assigned to the DataInput element, through the corresponding Data Associations.

Note that XPD L Formal Parameters have a Required attribute that maps to the BPMN 2.0 Data Input's "optional" but the default is inverted meaning the to achieve the BPMN 2.0 default behavior Required must be specified and must be

set to true.

### 7.1.10.6.Data Output

A Data Output is a visualization of one outgoing Formal Parameter of a Process. There may be multiple Data Outputs associated with a Process just as there may be more than one Formal Parameter. Data Outputs may appear in a Process diagram to show the outputs to the Process as a whole and may be denoted as the outputs of Activities by Data Associations.

- Data Inputs have the same notation as Data Objects, except MUST contain a small, filled block arrow
- Data Inputs MUST NOT have outgoing Data Associations except if the process they belong to is being called from another, external process. In this case a Data Association MAY be used to visualize where the Data Output is used by the calling process.

	Description
FormalParameterRef	IdRef of FormalParameter whose Mode="OUT". <i>Note:</i> Whilst there may be zero or more formal parameter references this is for purely historic reasons to mirror DataFieldRef in DataObjects, which is there to support BPMN 1.2 style DataObjects. Exactly one formal parameter reference is the expected usage.
Id	Id of the DataInput element. Type is xpdId.
Name	(optional) Name is an attribute that is a human readable name of the object.
NodeGraphicsInfos	Graphical information about flow object in the diagram.
Object	Used to associate categories and documentation with flow object.
State	(optional) Indicates the condition of the Data Object at this point in the Process. Multiple Data Objects with the same name MAY share the same state within one Process.

A Data Output element that references a Formal Parameter marked with IsArray="true" has to be visualized differently, compared to single *instance* data structures.



Single instance    Collection

### BPMN 2.0 compatibility

In BPMN 2.0, Data Outputs have an "optional" attribute that defines if a Data Output is valid even if the state is "unavailable". The default value is false. If the value of this attribute is true, then the execution of the Activity will not begin until a value is assigned to the DataOutput element, through the corresponding Data Associations.

Note that XPD L Formal Parameters have a Required attribute that maps to the BPMN 2.0 Data Output's "optional" but the default is inverted meaning the to achieve the BPMN 2.0 default behavior Required must be specified and must be set to true.

### 7.1.10.7.Data Associations

Data Associations are used to visualize the movement of data between the various visualizations of data (Data Objects, Data Stores, Data Inputs and Data Outputs), and inputs and outputs of Activities, Processes, and GlobalTasks. *Tokens* do not flow along a Data Association, and as a result they have no direct effect on the flow of the Process.

The purpose of Data Associations is purely visual and should not be confused with the execution semantics of mapping data into and out of Activities. The Data Association may however hold a reference to the Actual Parameter that defines the execution semantics. Refer to section 7.1.6.1 for execution semantics.

In XPD L a Data Association is a child of either a Process or ActivitySet reflecting its independence from source and target elements. Data Association elements have one or more sources and a target. In the simplest cases:

- The Data Association has just one source ; and

- The Data Field definitions of the source and target have the same structure/type.

If this is *not* so then:

- The Actual Parameter referred to by the Data Association MUST have EITHER a FormalExpression OR an Assignment that transforms the source structure/type into the target structure/type.

**Data Associations** are visually represented in the diagram by using the Association connector style:



A Data Association used to show the Outputs and Inputs of Activities:

	Description
ActualParameterRef	(optional) Defines Actual Parameter that holds the execution semantics for this data association. Type is xpd:IdRef.
ConnectorGraphicsInfo	(optional) Graphical information about flow object in the diagram.
Description	(optional) Longer textual description of the object than available in the name.
Extended Attributes	(optional) Vendor or user defined extension information.
From	Data aware element that is the source for this data association.
Id	Id of the DataInput element. Type is xpd:Id.
Name	(optional) Name is an attribute that is a human readable name of the object.
Object	(optional) Used to associate categories and documentation with flow object.
To	Data aware element that is the target for this data association.

### *Mapping of Data Inputs and Outputs to specific Activity Implementations*

In general, each Activity has one or more Formal Parameters each defining a data structure and which together represent the input output specification for the Activity. Data instances (Data Fields, Process Formal Parameters or Data Store References) are mapped to these by Actual Parameters. Refer to section 7.1.6.1.

The following describes the mapping of data inputs and outputs to the specific Activity implementations that apply further restrictions:

### **Service Task Mapping**

There is a single Formal Parameter element that has a data type equivalent to each parameter to the service being invoked. For example, if the service definition is made in a WSDL file there is one formal parameter for each WSDL part and the type of the formal parameter is the same as that of the part.

To map relevant data into or out of these messages refer to section 7.1.6.1: Formal-actual parameter mapping.

### **BPMN 2.0 compatibility**

BPMN 2.0 expects to find zero or one IN parameters and zero or one OUT parameters, which is consistent with the widely used document-literal style of web services.

## **7.2. Package Definition**

It is possible to define several processes within one package, which may share the same tools and participants. We recommend creating one package per business process which should contain all the necessary processes as well as all the associated tools and participants, although it is not required. It is also possible to define just parts of one process definition or common parts of several processes within one package (e.g. a participant list or an application list).

The details of Package are all in PackageType. The xml included in Package checks for referential integrity: it makes sure that elements that refer to other elements using id in fact refer to elements that exist.

```
<xsd:element name="Package" type="xpd:PackageType">
  <!-- checks that process id referred to by pool exists -->
```

```

<xsd:key name="ProcessIds.Package">
  <xsd:selector xpath="//xpd:WorkflowProcess | //xpd:ActivitySet"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="PoolProcessIdRef.Package" refer="xpd:ProcessIds.Package">
  <xsd:selector xpath="//xpd:Pool"/>
  <xsd:field xpath="@Process"/>
</xsd:keyref>
<!-- checks that process id referred to by subflow exists (must be top-level, not an activityset) -->
<xsd:key name="ProcessIdsTopLevel.Package">
  <xsd:selector xpath="//xpd:WorkflowProcess"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="SubFlowIdRef.Package" refer="xpd:ProcessIdsTopLevel.Package">
  <xsd:selector xpath="//xpd:SubFlow"/>
  <xsd:field xpath="@Id"/>
</xsd:keyref>
<!-- checks that start activityset referred to by subflow exists (note: incomplete test, does not constrain to process specified by
subflow) -->
<xsd:key name="ActivitySetIds.Package">
  <xsd:selector xpath="//xpd:ActivitySet"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="SubFlowStartActivitySetIdRef.Package" refer="xpd:ActivitySetIds.Package">
  <xsd:selector xpath="//xpd:SubFlow"/>
  <xsd:field xpath="@StartActivitySetId"/>
</xsd:keyref>
<!-- checks that start activity referred to by subflow exists (note: incomplete test, does not constrain to process specified by
subflow) -->
<xsd:key name="ActivityIds.Package">
  <xsd:selector xpath="//xpd:Activity"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="SubFlowStartActivityIdRef.Package" refer="xpd:ActivityIds.Package">
  <xsd:selector xpath="//xpd:SubFlow"/>
  <xsd:field xpath="@StartActivityId"/>
</xsd:keyref>
<!-- checks that activity referred to by taskreference exists (note: may be incomplete test, does not constrain to same process
that contains the task) -->
<xsd:keyref name="TaskReferenceTaskRef.Package" refer="xpd:ActivityIds.Package">
  <xsd:selector xpath="//xpd:TaskReference"/>
  <xsd:field xpath="@TaskRef"/>
</xsd:keyref>
<!-- checks that lane id referred to by nodegraphicsinfo exists -->
<xsd:key name="LaneIds.Package">
  <xsd:selector xpath="//xpd:Lane"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="NodeGraphicsInfoLaneIdRef.Package" refer="xpd:LaneIds.Package">
  <xsd:selector xpath="//xpd:NodeGraphicsInfo"/>
  <xsd:field xpath="@LaneId"/>
</xsd:keyref>
<!-- checks that source and target referred to by messageflow exists (note: incomplete test, does check that source/target are, or
are in, separate pools) -->
<xsd:key name="PoolAndActivityIds.Package">
  <xsd:selector xpath="//xpd:Pool | //xpd:Activity"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:keyref name="MessageFlowSourceRef.Package" refer="xpd:PoolAndActivityIds.Package">
  <xsd:selector xpath="//xpd:MessageFlow"/>
  <xsd:field xpath="@Source"/>
</xsd:keyref>
<xsd:keyref name="MessageFlowTargetRef.Package" refer="xpd:PoolAndActivityIds.Package">
  <xsd:selector xpath="//xpd:MessageFlow"/>
  <xsd:field xpath="@Target"/>
</xsd:keyref>
</xsd:element>

```

Schema removed, except for constraints.

Table needs to be updated for BPMN2.0 changes..

	Description
--	-------------

	Description
Applications	A list of Application Declarations. See section 7.3.
Artifacts	A list of Artifacts that can be linked to the existing Flow Objects through Associations. See sections 6.4.7. and 7.1.9.
Associations	A list of Associations which associate information and Artifacts with Flow Objects. See section 7.10.
ConformanceClass	Structural restriction on process definitions in this package. See section 7.2.3.
DataFields	A list of Relevant data fields defined for the package. See section 7.12.
DataStores	A list of DataStore elements.
ExtendedAttributes	A list of vendor-defined extensions that may be added to the package. See section 7.1.1.
ExternalPackages	Reference to another Package definition defined in a separate document.
GlobalActivities	List of Task Activities that are global.
Id	Used to identify the package.
Language	This holds the code for the language in which text is written as specified by ISO 639-2. Optionally this may be suffixed with a country code as specified by ISO 3166 to permit distinction between national dialects of the given language. The default is 'en_US'.
MessageFlows	A list of MessageFlows which go between Pools or activities in two pools. See section 7.8.
Name	Text. Used to provide a name for the Package.
PackageHeader	A set of elements specifying package characteristics.
Participants	A list of resources used in implementing processes in the package. See section 7.11.
PartnerLinkTypes	Partner link types for this package. See section 7.8.1.
Pools	A list of Pools for the Package. See section 7.4.
QueryLanguage	A Language MAY be provided so that the syntax of queries used in the Diagram can be understood. [Editors Note: Is this different than Scripting Language? TBD by BPMN.]
RedefinableHeader	A set of elements and attributes used by both the Package and Process definitions.
Script	Identifies the scripting language used in expressions.
TypeDeclarations	A list of Data Types used in the package. See section 7.13.
Processes	A list of the Processes that comprise this package. See section 7.5.

Table : Package Definition

### 7.2.1. Package definition Header

The package definition header keeps all information central to a package such as XPD L version, source vendor id, etc.

	Description
CostUnit	Units used in Simulation Data (Usually expressed in terms of a currency). The currency codes specified by ISO 4217 are recommended.
Created	Creation date of Package Definition. Should be stored in either the Basic or Extended forms specified by ISO 8601. For example: 1985-04-12T10:15:30Z is the extended form of the 3:30 pm on the 12th April 1985 GMT.
Description	Textual description of the package.
Documentation	Operating System specific path- and filename of help file/description file.



	Description
LayoutInfo	All co-ordinates (in NodeGraphicsInfos) have origin of 'top-left, relative to parent container'. Co-ordinate units are in pixels. However it would be nice to give other applications a hint as to the scale of a 'pixel' when the XPDL file was saved (i.e. the XPDL writer specifies co-ordinates and sizes in pixels but can also specify 'pixels to the millimeter' - the reading application can then, if it wishes, take this into account and scale to its pixel size appropriately). See PixelsPerMillimeter below.
ModificationDate	This defines the date on which the Diagram was last modified (for this Version). Should be stored in either the Basic or Extended forms specified by ISO 8601. For example: 1985-04-12T10:15:30Z is the extended form of the 3:30 pm on the 12th April 1985 GMT.
PixelsPerMillimeter	The default unit should be Pixels. The transformation of whatever measurement unit is used internally is left up to the implementing tool.
PriorityUnit	A text string with user defined semantics.
Vendor	Defines the origin of this model definition and contains vendor's name, vendor's product name and product's release number.
VendorExtensions	List of extensions by vendors. There is a vendor extension entry for each tool that provides extensions in this XPDL content.
XPDLVersion	Version of this specification. The current value, for this specification, is "2.2".

Table : Package Definition Header – Attributes

**7.2.1.1. Vendor extensions**

Vendor extension is used for vendors to define extensions and provide a schema and a description for the extensions. For details, see section 7.1.3.2 Namespace Qualified Extensions.

	Description
ToolId	Identification of the tool adding this extension. It is the same value used in NodeGraphicsInfo ToolId. This value may be a URI.
SchemaLocation deprecated: see next	A URI indicating the location of the schema that can be used to validate the XPDL containing the extensions for the tool.
SchemaNamespace	The URI of the namespace for the vendor extension. The XSD for this URI MAY be specified by the xsi:schemaLocation as shown in the example below.
ExtensionDescription	A URI indicating the location of a document describing the extensions provided by the schema in schemaLocation.

Table : Vendor Extension -- Attributes

**7.2.1.2. Example of specifying the vendor extension's schema location**

Vendor extensions can specify the location of the schema to permit tools to validate the XPDL containing the extensions by using the schemaLocation attribute as follows:

```
<xpdL2:Package xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:vendorExt="http://www.vendor.com/vendorExt"
  xmlns:xpdL2="http://www.wfmc.org/2004/XPDL2.0alpha"
  xsi:schemaLocation="xpdL2 http://www.wfmc.org/standards/docs/TC-1025_bpmnxpdL_24.xsd vendorExt
  http://www.vendor.com/VendorExtensions1.xsd"
  Id="_pTCasJIFEdye79YtjKhBOQ" Name="XPDL21Package">
```

In this example the vendor extension element would specify its SchemaNamespace attribute as:

```
http://www.vendor.com/vendorExt
```

### 7.2.2. Redefinable Header

The redefinable header covers those header attributes that may be defined in the definition header and may be redefined in the header of any process definition. In case of redefinition, the scope rules hold.

	Description
Author	Name of the author of this package definition.
Codepage	The codepage used for the text parts. If codepage is omitted, then UTF-8 is assumed.
Countrykey	Country code based on ISO 3166. It could be either the three digits country code number, or the two alpha characters country codes.
PublicationStatus	Status of the Process Definition. UNDER_REVISION RELEASED UNDER_TEST
Responsible(s)	Participant, who is responsible for this process; the supervisor during run time. Link to entity participant. Participant, who is responsible for this workflow of this Model definition (usually an Organisational Unit or a Human). It is assumed that the responsible is the supervisor during run time. Default: Initiating participant.
Version	Version of this Package Definition.

*Table : Redefinable Header*

### 7.2.3. Conformance Class Declaration

The conformance class declaration allows description of the conformance class to which the definitions in the model definition are restricted. The specified class applies to all the contained process definitions, unless it is re-defined locally at the process definition level.

There are two independent categories defined for conformance: Graph Conformance and BPMN Model Portability conformance.

#### 7.2.3.1. Graph Conformance

The following Graph Conformance Classes are defined at the package level:

- NON-BLOCKED
  - There is no restriction for this class.
- LOOP-BLOCKED
  - The Activities and Transitions/SequenceFlows of a Process Definition form an acyclic graph (or set of disjoint acyclic graphs).
- FULL-BLOCKED
  - For each join (or respectively split) there is exactly one corresponding split (or respectively join) of the same kind. In an AND split no conditions are permitted; in a XOR split an unconditional or OTHERWISE Transition is required if there is a Transition with a condition (i.e. an undefined result of transition evaluation is not permitted).

#### 7.2.3.2. BPMN Model Portability Conformance

##### 7.2.3.2.1. Overview

BPMN can be used for both "abstract" activity flow modeling and for complete executable design. Many tools, however, make use of BPMN for the abstract modeling but add executable detail in tool-specific activity properties. One goal of XPDL is to promote portability of abstract activity flow models between tools. This requires

separating the elements and attributes of BPMN related to activity flow modeling from those related to executable design. The BPMN1.x spec does not define this separation, but XPDL2.1 does, in the form of BPMN Model Portability conformance classes. Conformance subclasses for the Process Modeling part of BPMN2.0 have also been specified in the official OMG specification and included in XPDL2.2. We include a discussion of this work in section 7.2.3.4 .

In broad terms, the "abstract model" elements are those that represent BPMN constructs that are printable in the business process diagram, such as those defining the flow object type or subtype (e.g., looping User task, collapsed subprocess, exclusive gateway, timer event), including only attributes specifying the subtype, label (Name attribute), and unique identifiers for the object itself and pointers to other identifiers in the diagram. Elements and attributes representing data, messages, or other implementation detail are omitted from the abstract process model. In other words, the model describes the "what" and the "when" of process activity flow, but not the "how" of flow object implementation.

XPDL defines three Model Portability conformance classes, SIMPLE, STANDARD, and COMPLETE. A modeling tool asserting compliance to one of these classes means that the tool can import and understand all parts of a serialized BPMN instance conformant to the class. All three classes ignore vendor specific extensions via the use of Extended Attributes or user name spaces. They differ only in the set of abstract modeling elements supported.

The SIMPLE class includes the following BPMN objects: task, collapsed subprocess, gateway (exclusive data-based, parallel), None start and None end events, pool, lane, data object, text annotation, sequence flow (uncontrolled, conditional, default), and association.

The STANDARD class includes the following BPMN objects: task (task type User, Service, Send, Receive); collapsed and expanded subprocess, looping or multi-instance activity, gateway (inclusive, exclusive data-based, exclusive event-based, parallel), start events (None, message, timer), catching intermediate events in sequence flow (timer, message), throwing intermediate events in sequence flow (message), attached intermediate events (timer, message, error), end events (None, error, message, terminate), pool, lane, data object, text annotation, sequence flow (uncontrolled, conditional, default), and association.

The COMPLETE class includes all task types, all event types, and all gateway types described by BPMN 1.1, message flow, transactional subprocess, and ad hoc subprocess.

An XSL transform (XSLT) applied to an XPDL instance identifies the claimed conformance class by looking at the conformance attribute. Elements and attributes not belonging to the class are reported. Additional validation rules check for structural integrity.

#### 7.2.3.2.2. *BPMN Model Portability Conformance Class Summary*

The following BPMN Model Portability Conformance classes are defined at the package level:

- NONE
  - The document does not support BPMN.
- SIMPLE
  - The document conforms to the requirements for portability of models containing a simple set of BPMN elements.
- STANDARD
  - The document conforms to the requirements for portability of models containing a standard set of BPMN elements.
- COMPLETE
  - The document conforms to the requirements for portability of models containing the full set of BPMN elements.

We will provide tools for validating BPMN abstract model portability conformance classes.

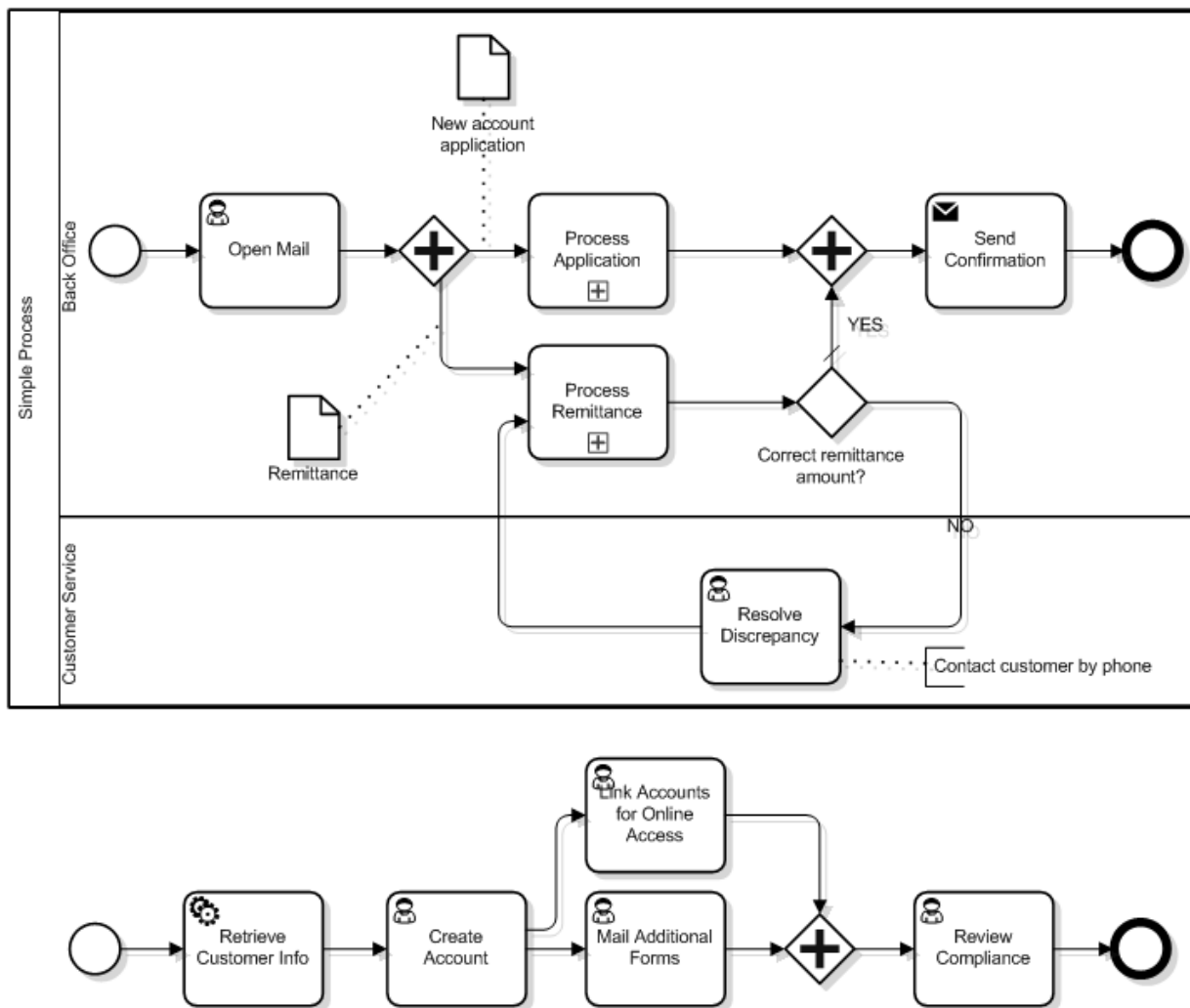


Figure 7.: Simple Class

We envision other BPMN portability conformance classes, and other levels of abstract model portability conformance may be introduced in the future.

### 7.2.3.3. Conformance Schema

	Description	
GraphConformance	FULL-BLOCKED	The network structure is restricted to proper nesting of SPLIT/JOIN and loops.
	LOOP-BLOCKED	The network structure is restricted to proper nesting of loops.
	NON-BLOCKED	There is no restriction on the network structure. This is the default.

	Description	
BPMNModelPortabilityConformance	NONE	The document is not BPMN compliant.
	SIMPLE	The document conforms to the requirements for portability of models containing a simple set of BPMN elements.
	STANDARD	The document conforms to the requirements for portability of models containing the standard set of BPMN elements.
	COMPLETE	The document conforms to the requirements for portability of models containing the full set of BPMN elements.

Table : Conformance Class Declaration

#### 7.2.3.4. BPMN2.0 Conformance Subclasses

XPDL2.2 has a new set of Conformance classes, to correspond with the BPMN2.0 conformance subclasses.

As an alternative to full Process Modeling Conformance, there are three conformance sub-classes defined:

- DESCRIPTIVE
- ANALYTIC
- COMMONEXECUTABLE

DESCRIPTIVE is concerned with visible elements and attributes used in high-level modeling. It should be comfortable for analysts who have used BPA flowcharting tools.

ANALYTIC contains all of DESCRIPTIVE and in total about half of the constructs in the Full Conformance Class. It is based on experience gathered in BPMN training and an analysis of user-patterns in the Department of Defense Architecture Framework and planned standardization for that framework. Both DESCRIPTIVE and ANALYTIC focus on visible elements and a minimal subset of supporting attributes/elements.

COMMONEXECUTABLE focuses on what is needed for executable process models.

Elements and attributes not in these sub-classes are contained in the full Process Modeling Conformance class.

For a tool to claim support for a sub-class the following criteria must be satisfied:

- All the elements in the sub-class must be supported.
- For each element, all the listed attributes must be supported.
- In general, if the sub-class doesn't mention an attribute and it is not required by the schema then it is not in the subclass. Exceptions to this rule are noted.

The conformance sub-class elements are as follows:

##### 7.2.3.4.1. DESCRIPTIVE

ELEMENT	ATTRIBUTES
participant (pool)	id, name, processRef
laneSet	id, lane with name, childLaneSet, flowElementRef
sequenceFlow (unconditional)	id, name, sourceRef, targetRef
messageFlow	id, name, sourceRef, targetRef
exclusiveGateway	id, name
parallelGateway	id, name
task (None)	id, name
userTask	id, name
serviceTask	id, name

subProcess (expanded)	id, name, flowElement
subProcess (collapsed)	id, name, flowElement
CallActivity	id, name, calledElement
DataObject	id, name
TextAnnotation	id, text
association/dataAssociation***	id, name, sourceRef, targetRef, associationDirection*
dataStoreReference	id, name, dataStoreRef
startEvent (None)	id, name
endEvent (None)	id, name
messageStartEvent	id, name, messageEventDefinition
messageEndEvent	id, name, messageEventDefinition
timerStartEvent	id, name, timerEventDefinition
terminateEndEvent	id, name, terminateEventDefinition
documentation**	text
Group	id, categoryRef

\* associationDirection not specified for dataAssociation

\*\* documentation is not a visible element. It is an attribute of most elements.

\*\*\* dataAssociation is ABSTRACT: dataInputAssociation and dataOutputAssociation will appear in the XML serialization. These both have required attributes[sourceRef and targetRef] which refer to data-aware elements. To be consistent with the metamodel, this will require the following additional elements: ioSpecification, inputSet, outputSet, dataInput, dataOutput. When a BPMN editor draws a dataAssociation to an Activity or Event it should generate this supporting invisible substructure. Otherwise, the metamodel would have to be changed to make sourceRef and targetRef optional or allow reference to non-data-aware elements, e.g. Activity and Event.

#### 7.2.3.4.2. ANALYTIC

All the elements in DESCRIPTIVE plus:

ELEMENT	ATTRIBUTES
sequenceFlow (conditional)	id, name, sourceRef, targetRef, conditionExpression*
sequenceFlow (default)	id, name, sourceRef, targetRef, default**
sendTask	id, name
receiveTask	id, name
Looping Activity	standardLoopCharacteristics
MultiInstance Activity	multiInstanceLoopCharacteristics
exclusiveGateway	Add default attribute
inclusiveGateway	id, name, eventGatewayType
eventBasedGateway	id, name, eventGatewayType
Link catch/throw IE***	Id, name, linkEventDefinition
signalStartEvent	id, name, signalEventDefinition
signalEndEvent	id, name, signalEventDefinition

Catching message IE***	id, name, messageEventDefinition
Throwing message IE	id, name, messageEventDefinition
Boundary message IE	id, name, attachedToRef, messageEventDefinition
Non-int**** Boundary messageIE	id, name, attachedToRef, cancelActivity=false, messageEventDefinition
Catching timer IE	id, name, timerEventDefinition
Boundary timer IE	id, name, attachedToRef, timerEventDefinition
Non-int Boundary timer IE	id, name, attachedToRef, cancelActivity=false, timerEventDefinition
Boundary error IE	id, name, attachedToRef, errorEventDefinition
errorEndEvent	id, name, errorEventDefinition
Non-int Boundary escalation IE	id, name, attachedToRef, cancelActivity=false, escalationEventDefinition
Throwing escalation IE	id, name, escalationEventDefinition
escalationEndEvent	id, name, escalationEventDefinition
Catching signal IE	id, name, signalEventDefinition
Throwing signal IE	id, name, signalEventDefinition
Boundary signal IE	id, name, attachedToRef, signalEventDefinition
Non-int Boundary signal IE	id, name, attachedToRef, cancelActivity=false, signalEventDefinition
conditionalStartEvent	id, name, conditionalEventDefinition
Catching conditional IE	id, name, conditionalEventDefinition
Boundary conditional IE	id, name, conditionalEventDefinition
Non-int Boundary conditional IE	id, name, cancelActivity=false, conditionalEventDefinition
message*****	id, name, add messageRef attribute to messageFlow

\* conditionExpression allowed only for sequenceFlow out of gateways, may be null.

\*\* default is an attribute of a sourceRef (exclusive or inclusive) DecisionGateway.

\*\*\* IE= intermediateEvent

\*\*\*\* Non-int=nonInterrupting

\*\*\*\*\*Note that messageRef, an attribute of various message Events, is optional and not in the sub-class.

#### 7.2.3.4.3. *Common Executable*

This class is intended for modeling tools that can emit executable models.

Data type definition language MUST be XML Schema. Service Interface definition language MUST be WSDL. Data access language MUST be XPath.

ELEMENT	ATTRIBUTES
sequenceFlow (unconditional)	id, (name), sourceRef*, targetRef**
sequenceFlow (conditional)	id, name, sourceRef*, targetRef**, conditionExpression***
sequenceFlow (default)	id, name, sourceRef*, targetRef**, default****
subProcess (expanded)	id, name, flowElement, loopCharacteristics, boundaryEventRefs
exclusiveGateway	id, name, gatewayDirection (only converging and diverging), default
parallelGateway	id, name, gatewayDirection (only converging and diverging)

startEvent (None)	id, name
endEvent (None)	id, name
eventBasedGateway	id, name, gatewayDirection, eventGatewayType
userTask	id, name, renderings, implementation, resources, ioSpecification, dataInputAssociations, dataOutputAssociations, loopCharacteristics, boundaryEventRefs
serviceTask	id, name, implementation, operationRef, ioSpecification, dataInputAssociations, dataOutputAssociations, loopCharacteristics, boundaryEventRefs
callActivity	id, name, calledElement, ioSpecification, dataInputAssociations, dataOutputAssociations, loopCharacteristics, boundaryEventRefs
dataObject	id, name, isCollection, itemSubjectRef
textAnnotation	id, text
dataAssociation	id, name, sourceRef, targetRef, assignment
messageStartEvent	id, name, messageEventDefinition (either ref or contained), dataOutput, dataOutputAssociations
messageEndEvent	id, name, messageEventDefinition, (either ref or contained), dataInput, dataInputAssociations
terminateEndEvent	(Terminating trigger in combination with one of the other end events)
Catching message IE	id, name, messageEventDefinition (either ref or contained), dataOutput, dataOutputAssociations, <b>Correlation(waiting for J-384 to get exact attribute and class list)</b>
Throwing message IE	id, name, messageEventDefinition (either ref or contained), dataInput, dataInputAssociations
Catching timer IE	id, name, timerEventDefinition (contained)
Boundary error IE	id, name, attachedToRef, errorEventDefinition, (contained or referenced), dataOutput, dataOutputAssociations
<b>Supporting classes</b>	
StandardLoopCharacteristics	id, loopCondition
MultiInstanceLoopCharacteristics	id, isSequential, loopDataInput, inputDataItem
Rendering	
Resource	id, name
ActivityResource	id, resourceRef, resourceAssignmentExpression
InputOutputSpecification	id, dataInputs, dataOutputs
DataInput	id, name, isCollection, itemSubjectRef
DataOutput	id, name, isCollection, itemSubjectRef
ItemDefinition	id, structure or import*****
Operation	id, name, inMessageRef, outMessageRef, errorRefs
Message	id, name, structureRef
Error	id, structureRef
Assignment	id, from, to*****
MessageEventDefinition	id, messageRef, operationRef



TerminateEventDefinition	id
TimerEventDefinition	id, timeDate

\*Multiple outgoing connections are only allowed for diverging gateways.

\*\*Multiple incoming connections are only allowed for converging gateways.

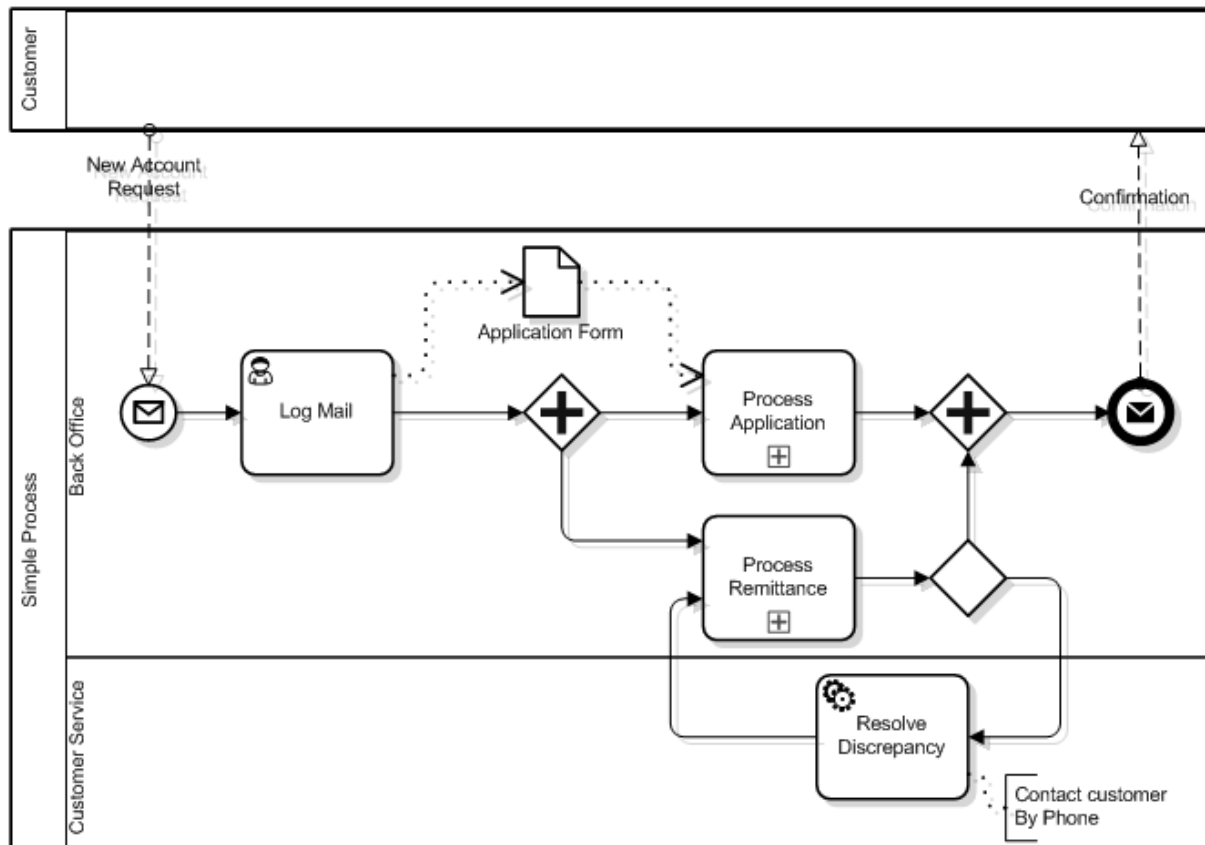
\*\*\* conditionExpression allowed only for sequenceFlow out of gateways, may be null.

\*\*\*\* default is an attribute of a sourceRef (exclusive or inclusive) DecisionGateway.

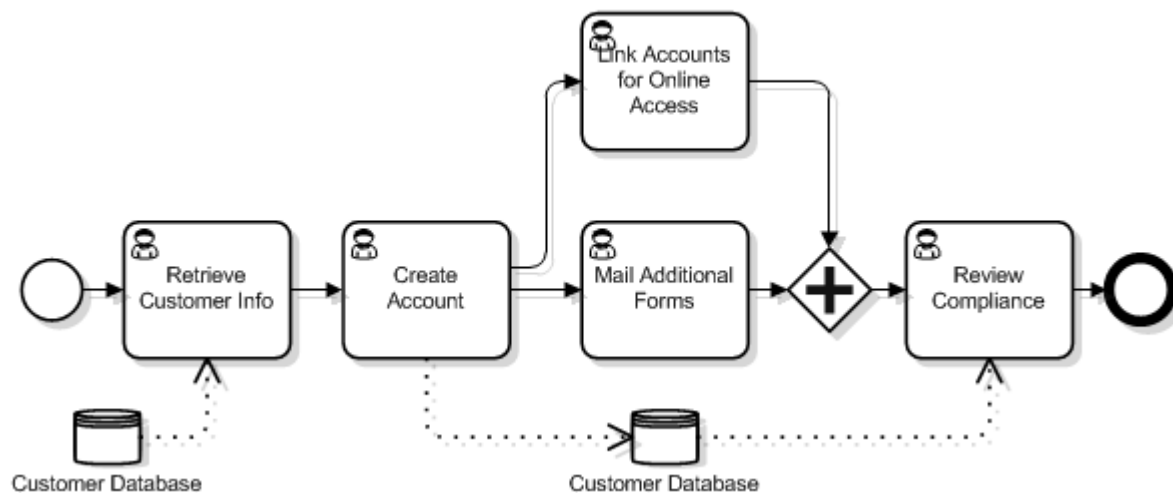
\*\*\*\*\* Structure must be defined by an XSD Complex Type

### 7.2.3.5. Examples

#### 7.2.3.5.1. DESCRIPTIVE



## Expansion of Process Application



### 7.2.4. Script

The Script element identifies the scripting language used in XPDL expressions. A text expression may be used wherever an element is of type `xsd:string`. One could, for example, use an expression within Cost elements.

An expression composed of formatted XML (e.g., MathML) may be used within the ActualParameter or Expression element (used within a TransitionCondition).

	Description
Type	Identifies the scripting language used in expressions. For consistency across implementations, when specifying a standard scripting language, it is recommended that the Type be selected from the following strings: <code>text/javascript</code> , <code>text/vbscript</code> , <code>text/tcl</code> , <code>text/ecmascript</code> , <code>text/xml</code> , <code>application/xslt+xml</code> . For BPMN2.0 the default scripting language is XPATH.
Version	This is the version of the scripting language.
Grammar	This is a reference to a document that specifies the grammar of the language. It could be, for example, an XML schema, a DTD, or a BNF.

Table : Script

### 7.2.5. External Package

External package reference allows referencing definitions in another Package definition or in other systems providing an Interface to the Process or Management system (e.g. a legacy Organisation Description Management Tool).

The ExternalPackages element allows XPDL modelers to isolate common objects in packages that can be included in multiple models. The element holds a list of ExternalPackage elements each of which refer to another XPDL document, an external package. External packages are indistinguishable from any other package and can contain any well formed XPDL model. However, when they are included into another document as an external package, only some of the information in the external package will be examined and used. In particular, the package level and process level Applications and Participants and the processes in external package can be used within the containing document.

The schema describing XPDL states that the ExternalPackages element appears within the Package element and that it may contain zero or more ExternalPackage elements. The ExternalPackage elements have three attributes: an id, a

name, and a reference to the document to be included.

```

<xsd:element name="ExternalPackage">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xpd:ExtendedAttributes" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="href" type="xsd:string"/>
    <xsd:attribute name="Id" type="xsd:NMTOKEN"/>
    <xsd:attribute name="Name" type="xsd:string" use="optional"/>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="ExternalPackages">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xpd:ExternalPackage" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

```

	Description
ExtendedAttributes	Optional vendor-defined extensions to meet implementation needs. See section <a href="#">7.1.3</a> .
Href	A Model Identifier. Logical reference to a Model.
Id	The id of externally referenced package.
Name	The name given to the external package.

*Table : External Package Reference*

### 7.2.6. Example use of External Package

Here is an example extracted from a document generated by one company's XPD/BPMN design tool:

```

<ExternalPackages>
  <ExternalPackage Id="11" Name="Muses" href="C:\Work\External Packages\Muses.xpdl" />
  <ExternalPackage Id="12" Name="Stooges" href="C:\Work\External Packages\Stooges.xpdl" />
</ExternalPackages>

```

When an application or participant name is mentioned in a document there is nothing to indicate it originates in an external package, so some effort must be made to insure that those names are unique.

When a process Id is specified in a Subflow or TaskApplication element, the PackageRef attribute is used to specify which external package defines the process.

## 7.3. Application Declaration

Application declaration is a list of all applications/services or tools required and invoked by the processes defined within the process definition or surrounding package. Tools may be defined (or, in fact, just named). This means, that the real definition of the tools is not necessary and may be handled by an object manager. The reason for this approach is the handling of multi-platform environments, where a different program (or function) has to be invoked for each platform. XPD abstracts from the concrete implementation or environment (thus these aspects are not of interest at process definition time).

	Description
Description	Short textual description of the application.
ExtendedAttributes	Optional vendor-defined extensions to meet implementation needs. See section <a href="#">7.1.3</a> .
ExternalReference	A reference to an external specification of the application signature. See section <a href="#">7.1.5</a> .
FormalParameters	A list of parameters that are interchanged with the application via the invocation interface. See section <a href="#">7.1.4</a> .
Id	Used to identify the application definition.
Name	Text used to identify an application (may be interpreted as a generic name of the tool).
Type	There are a number of standard Application Types. See section <a href="#">7.3.2</a> .

*Table : Application Declaration*

### 7.3.1. Invocation Parameters

An Application declaration may have parameter definitions for the (invocation) parameters and also use them within other entities.

Copying the invocation IN is treated as one atomic operation. The same holds for restoring the invocation OUT. Between these two operations no assumption is made about concurrency behaviour.

### 7.3.2. Application Types

Application Type contains several pieces of information required by common applications such as calling an EJB component or invoking a WebService. Support for a particular application type is not mandatory for the engine, but if the engine makes use of the listed technologies the additional information should be provided by Application Type.

#### 7.3.2.1. EJB

Application type that defines information required to call a method of an EJB component. An EJB application has additional restrictions for formal parameters: there can be a maximum of one OUT formal parameter, and if it exists it has to be the last formal parameter, also there should be no INOUT formal parameters. With these restrictions IN formal parameters map directly to arguments of the method and the optional last OUT formal parameter becomes the return value of the method.

	Description
JndiName	JNDI name of the EJB.
HomeClass	Home class fully qualified name.
Method	Method that will be invoked.

*Table : EJB Application Type*

#### 7.3.2.2. POJO

Application type that defines information required to call method on local Java class. Formal Parameters restrictions are the same as for EJB application type.

	Description
Class	Fully qualified name of the class.
Method	Method that will be invoked.

*Table : POJO Application Type***7.3.2.3. XSLT**

Application that uses XSL transformation on formal parameters. The Application should have one IN and one OUT formal parameter, or if the transformation transforms the formal parameter into a document in the same schema the application can have one INOUT formal parameter.

	Description
Location	Location of the XSL transformation.

*Table : XSLT Application Type***7.3.2.4. Script**

Application that executes a script (expression) using formal parameters. The script should have access only to formal parameters of the application.

	Description
Expression	The script.

*Table : Script Application Type***7.3.2.5. WebService**

Application that invokes a Web Service. All IN formal parameters should be mapped into content of the input message; all OUT formal parameters should be mapped to parts of the output message.

	Description
WebServiceOperation	The web service operation used to invoke this application.
WebServiceFaultCatch	Provides a way to catch faults generated by the application.
InputMsgName	The name of inputMessage as defined in the WSDL which will help in uniquely identifying the operation to be invoked.
OutputMsgName	The name of outputMessage as defined in the WSDL which will help in uniquely identifying the operation to be invoked.

*Table : WebService Application Type***7.3.2.6. BusinessRule**

Application that invokes a Business Rule.

	Description
RuleName	Name of the Business Rule.
Location	Location of the Rule.

*Table : BusinessRule Application Type***7.3.2.7. Form**

The standard does not decide which kind of a form layout should be used. The Form Application Type defines a place where all form related information should be stored.

	Description
FormLayout	Optional description of form layout.

*Table : Form Application Type*

## 7.4. Swimlanes

BPMN uses the concept known as “swimlanes” to help partition and organize activities. It is possible that a BPMN Diagram may depict more than one private process, as well as the processes that show the collaboration between private processes or Participants. If so, then each private business process will be considered as being performed by a different Participant. Graphically, each Participant will be partitioned; that is, will be contained within a rectangular box call a “Pool.” Pools can have sub-Swimlanes that are called, simply, “Lanes.”

BPMN private processes correspond to XPDL processes. The BPMN term ‘participant’ is not the same as the XPDL ‘participant’ element.

For a general description of SwimLanes, Pools and Lanes refer to section 6.4.1.

### 7.4.1. Pool

A Pool represents a Participant in the Process. A Participant can be a specific business entity (e.g. a company) or can be a more general business role (e.g., a buyer, seller, or manufacturer). Graphically, a Pool is a container for partitioning a Process from other Pools, when modeling business-to-business situations, although a Pool need not have any internal details (i.e., it can be a “black box”).

**Note that the term Participant in the context of Pools is a BPMN concept that differs from the same term used in XPDL Participant Declaration, Participant Assignment and Performer expressions.**

#### 7.4.1.1. BPMN Graphics for Pools

- A Pool is a square-cornered rectangle that **MUST** be drawn with a solid single black line.
  - One, and only one, Pool in a diagram **MAY** be presented without a boundary. If there is more than one Pool in the diagram, then the remaining Pools **MUST** have a boundary.

Note: Some XPDL editors support the notion of multiple pages, where each page will have a background pool.



*Figure 7.: Pool*

To help with the clarity of the Diagram, A Pool will extend the entire length of the Diagram, either horizontally or vertically. However, there is no specific restriction to the size and/or positioning of a Pool. Modelers and modeling tools can use Pools (and Lanes) in a flexible manner in the interest of conserving the “real estate” of a Diagram on a screen or a printed page. A Pool acts as the container for the Sequence Flow between activities. The Sequence Flow can cross the boundaries between Lanes of a Pool, but cannot cross the boundaries of a Pool. The interaction between Pools is shown through Message Flow.

Another aspect of Pools is whether or not there is any activity detailed within the Pool. Thus, a given Pool may be shown as a “White Box,” with all details exposed, or as a “Black Box,” with all details hidden. No Sequence Flow is

associated with a “Black Box” Pool, but Message Flow can attach to its boundaries (see Figure below).

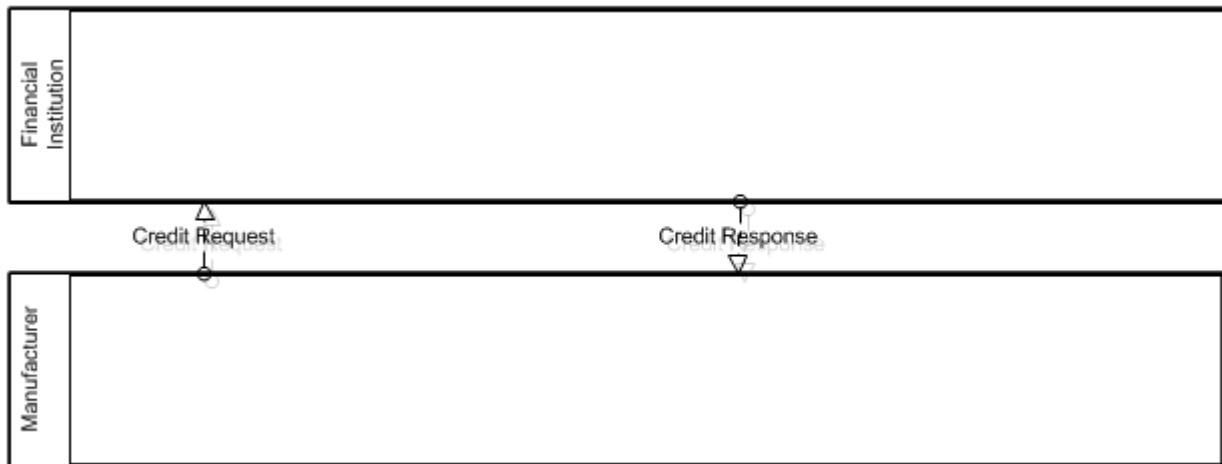


Figure 7.: Message Flow connecting to the boundaries of two Pools

For a “White Box” Pool, the activities within are organized by Sequence Flow. Message Flow can cross the Pool boundary to attach to the appropriate activity (see below).

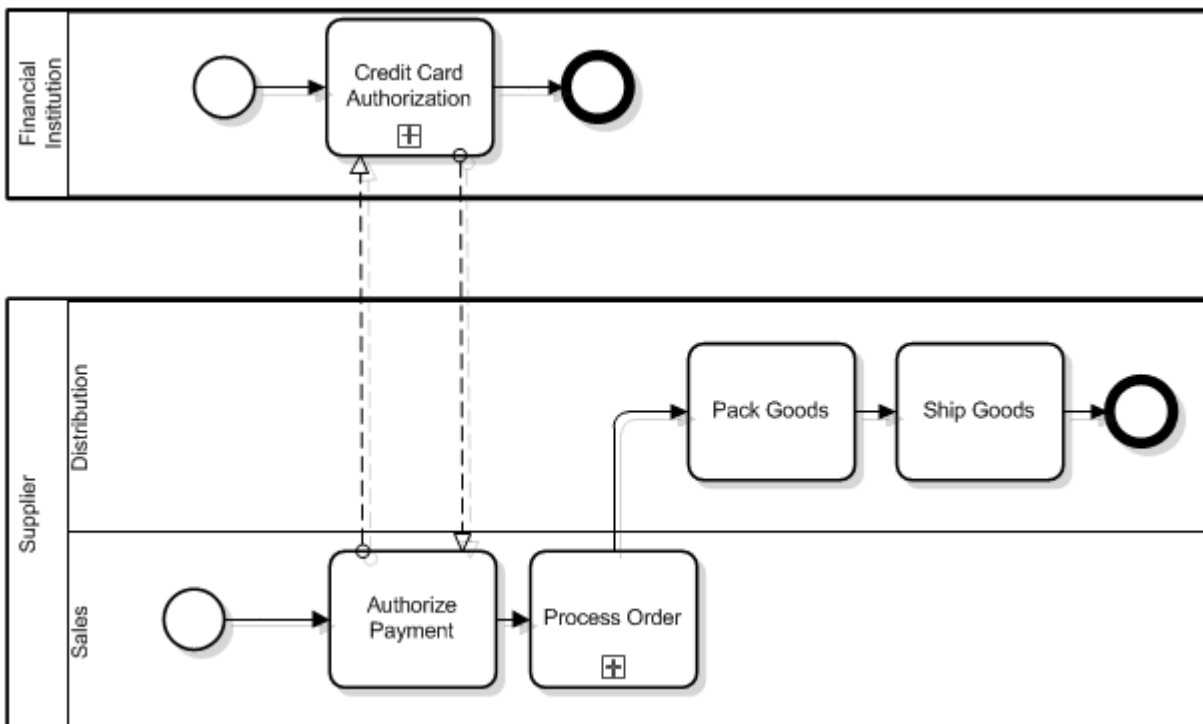


Figure 7.: Message Flow connecting to Flow Objects within two Pools

In BPMN1.x all Business Process Diagrams contain at least one Pool. See the following paragraph about BPMN2.0. In most cases, a BPD that consists of a single Pool will only display the activities of the Process and not display the boundaries of the Pool. Furthermore, a BPD may show the “main” Pool without boundaries. In such cases there can be, at most, only one invisibly-bounded pool in the diagram and the name of that pool can be the same as the diagram. [Note: In XPD the concept of “PACKAGE” is a generalization of a BPD. There is no necessary correspondence between the PACKAGE name and the Process or Pool name.] Consequently, the activities that represent the work performed from the point of view of the modeler or the modeler’s organization are considered “internal” activities and

need not be surrounded by the boundaries of a Pool, while the other Pools in the Diagram must have their boundary (see Figure below).

In BPMN2.0 there is no longer the concept of a BPD. A model with a single top-level process definition may not require a Pool at all. At the time of this writing the matter is being discussed in the Finalization Task Force.

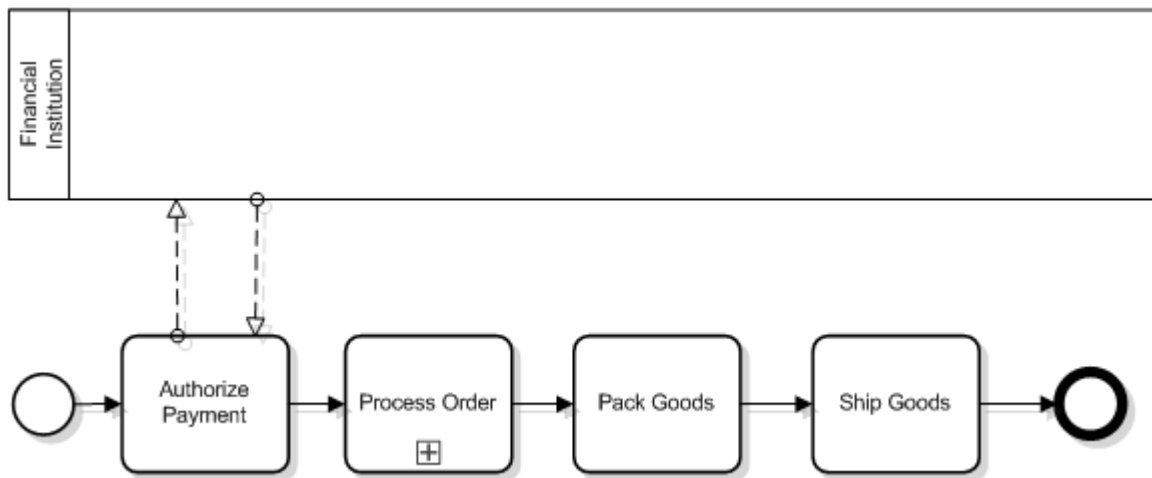


Figure 7.: Main (Internal) Pool without boundaries

#### 7.4.1.2. Schema for Pools

Schema in XSD file.

	Description
BoundaryVisible	This attribute defines if the rectangular boundary for the Pool is visible. Only one Pool on a page MAY have the attribute set to False.
Id	The id of the Pool.
Lanes	The lanes in the pool. See section 7.4.2.
MainPool	This attribute defines if the Pool is the “main” Pool or the focus of the diagram. Only one Pool in the Diagram MAY have the attribute set to True.
MultiInstance	Default: false. True means a Pool with a multiple participant.
Name	The name of the pool.
NodeGraphicsInfos	See section 7.1.1.
Object	See section 7.1.9.4
Orientation	HORIZONTAL   VERTICAL
Participant	The Modeler MAY define the Participant for a Pool. The Participant can be either a Role or an Entity. This defines the role that the Pool will play in a Diagram that includes collaboration. <b>Note that the term Participant in the context of Pools is a BPMN concept that differs from the same term used in XPDLP Participant Declaration, Participant Assignment and Performer expressions.</b> Within a Pool, the Lane organizes activities by role/entity & a performer expression identifies the Performer(s) (i.e. role/entity) that are performing the tasks in a Lane.
Process	
ProcessRef in BPMN	The Process attribute defines the Process that is contained within the Pool. Each Pool MAY have a Process.

Table : Pools



## 7.4.2. Lane

### 7.4.2.1. BPMN Graphics for Lanes

A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically (see Figure below) or horizontally (see Figure below). If the pool is invisibly bounded, the lane associated with the pool must extend the entire length of the pool. Text associated with the Lane (e.g., its name and/or any attribute) can be placed inside the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor. Our examples place the name as a banner on the left side (for horizontal Pools) or at the top (for vertical Pools) on the other side of the line that separates the Pool name; however, this is not a requirement.

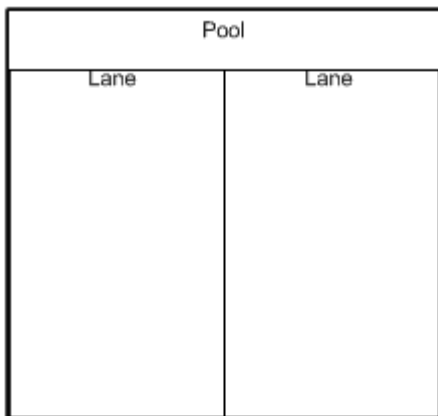


Figure 7.: Two Lanes in a Vertical Pool

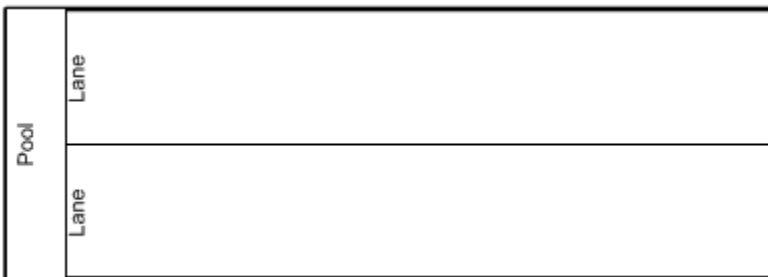


Figure 7.: Two Lanes in a Horizontal Pool

Lanes are used to organize and categorize activities within a Pool. The meaning of the Lanes is up to the modeler. BPMN does not specify the usage of Lanes. Lanes are often used for such things as internal roles (e.g., Manager, Associate), systems (e.g., an enterprise application), an internal department (e.g., shipping, finance), etc. In addition, Lanes can be nested (see Figure below) or defined in a matrix. For example, there could be an outer set of Lanes for company departments and then an inner set of Lanes for roles within each department.

Note: In XPD L we have added the **optional** Performers element that specifically designates a set of default performers for all TASKS in the Lane.

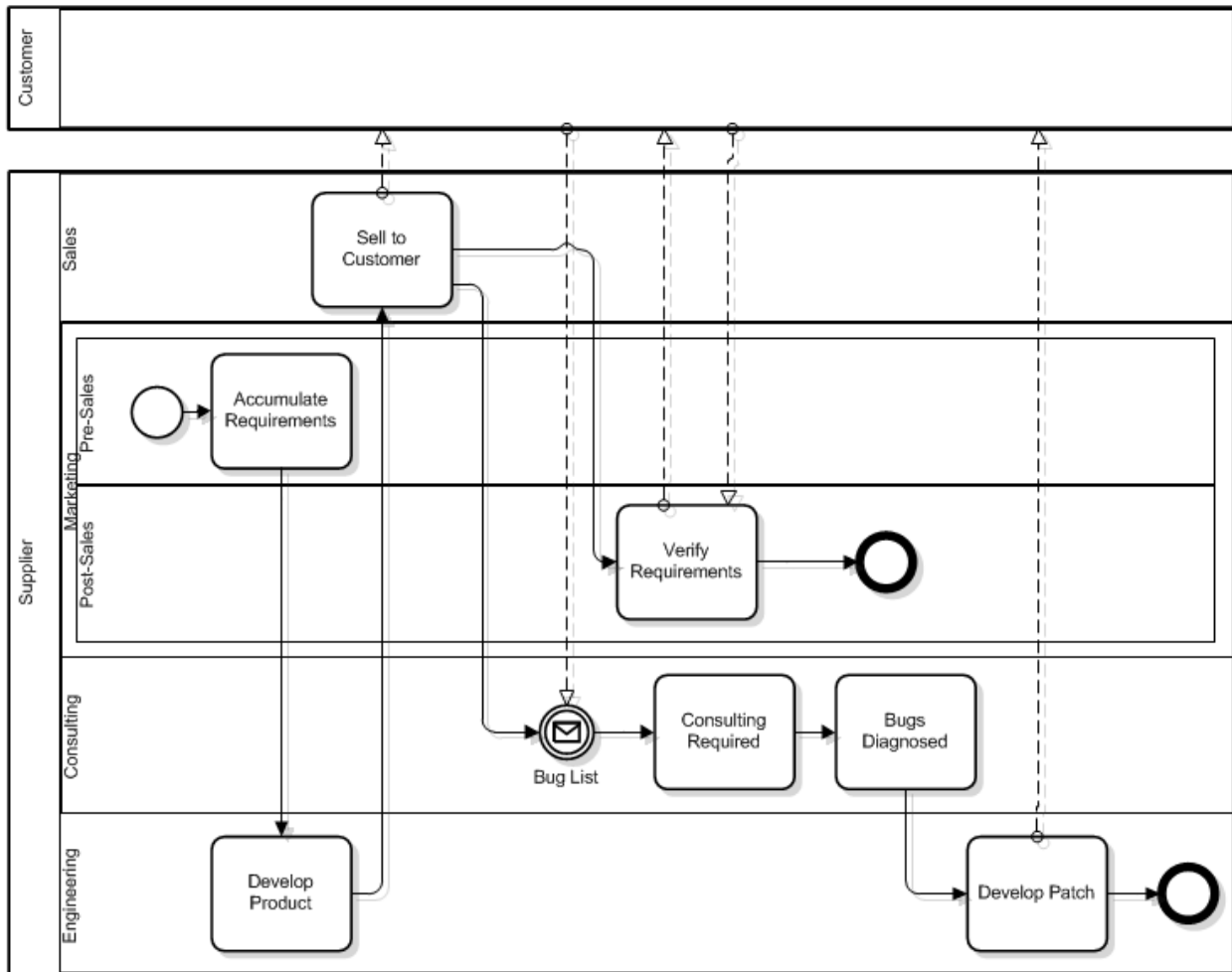


Figure 7.: An Example of Nested Lanes

#### 7.4.2.2. Schema for Lanes

	Description
Id	The id of the Lane.
NestedLane	This element identifies any Lanes that are nested within the current Lane.
Name	The name of the Lane.
NodeGraphicsInfos	See section 7.1.1.
Object	See section 7.1.9.4.
Performers	A Swim Lane in a Pool is often used to designate the default 'Role' required to perform any of the activities in the lane. This optional element provides the ability to specify the default performers for all activities in the lane. Any individual activity can override this with its own performer expression.

Table : Lane

## 7.5. Process Definition

The Process Definition defines the elements that make up a process. It contains definitions or declarations, respectively, for Activity and, optionally, for Transition, Application, and Process Relevant Data entities. Attributes may be specified for administration relevant data like author, and version; for runtime relevant data like priority; and for BPR and simulation relevant data.

BPMN Processes have similar attributes. A BPMN re-usable process is contained in a Pool which refers to the process via its Process attribute.

A Process may run as an implementation of an activity of type SubFlow; in this case parameters may be defined as attributes of the process. BPMN uses the term Sub-Process. See details in 7.6.5.4.2. Also see Semantics of Reusable Subprocess [BPMN perspective] in section 7.6.5.4.3. For details about the start and end of sub-processes see 7.6.5.4.5.

Note that XPDL also includes the notion of ActivitySet, described in the sequel (section 7.6.3: BlockActivity or EmbeddedSubProcess).

Where a process definition includes input parameters and is instantiated by means other than a SubFlow/subprocess call (for example by local event) the method for initializing any input parameters is locally defined. In such circumstances any relevant data field associated with the instantiated process definition, which is included within the parameter list will be initialized to the value specified in the “default value” (where specified). Where relevant data field is not passed as an input parameter, or initialized by “default value” the result is undefined. Similarly where a subflow terminates abnormally without returning out parameter values to the calling process, the result is undefined.

In general the scope of the defined entity identifier and name is the surrounding entity. The identifier is unique in this scope. For the Process identifier and name the scope is the surrounding Package.

When a process definition is instantiated it is necessary to determine which activity is the first (start) activity. There are a number of ways to do this.

- If there is only one activity that has no incoming transitions this is obvious.
- A single activity may have its StartActivity attribute set to true.
- The process attributes DefaultStartActivitySet and/or DefaultStartActivity may be present.
- The process invocation may specify the StartActivitySet and/or StartActivity.

Here we summarize the rules.

The following logic applies:

- a. Unless otherwise specified in the process invocation (see ProcessRef 7.6.5.4), DefaultStartActivitySet and/or DefaultStartActivity determine where the process will start executing.
- b. If present, DefaultStartActivitySetId must be the id of an Activity Set in the process
- c. If present, DefaultStartActivityId must be the id of a start activity
  - i. In the Default Activity Set if that's present
  - ii. In the top level process activities otherwise
- d. If DefaultStartActivitySetId is present but not DefaultStartActivityId it is assumed that DefaultStartActivitySetId has exactly one start activity
- e. If neither is present it is assumed that the top level activities contain exactly one start activity
- f. It is assumed that a process invocation can designate StartActivitySetId and StartActivityId and thereby control where process execution starts. In particular sub process invocation can contain the information (see ProcessRef 7.6.5.4).

**BPMN has a more complicated semantics for determining when and where a process starts. Refer to section 7.6.4.2 Start Event for a complete discussion.**

### 7.5.1. Schema

The details for WorkflowProcess are in ProcessType. The XML in WorkflowProcess enforces referential integrity.

```
<xsd:element name="WorkflowProcess" type="xpd:ProcessType">
  <xsd:key name="ActivitySetIds.WorkflowProcess">
```

```

        <xsd:selector xpath="/xpd:ActivitySets/xpd:ActivitySet"/>
        <xsd:field xpath="@Id"/>
    </xsd:key>
    <xsd:key name="ActivityIds.WorkflowProcess">
        <xsd:selector xpath="/xpd:Activities/xpd:Activity |
./xpd:ActivitySets/xpd:ActivitySet/xpd:Activities/xpd:Activity"/>
        <xsd:field xpath="@Id"/>
    </xsd:key>
    <!-- constrain to only activities in the top-level, not activitysets -->
    <xsd:key name="ActivityIdsTopLevel.WorkflowProcess">
        <xsd:selector xpath="/xpd:Activities/xpd:Activity"/>
        <xsd:field xpath="@Id"/>
    </xsd:key>
    <!-- constrain to only transitions in the top-level, not activitysets -->
    <xsd:key name="TransitionIdsTopLevel.WorkflowProcess">
        <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
        <xsd:field xpath="@Id"/>
    </xsd:key>
    <!-- check that specified default start activityset exists -->
    <xsd:keyref name="DefaultStartActivitySetIdRef.WorkflowProcess" refer="xpd:ActivitySetIds.WorkflowProcess">
        <xsd:selector xpath="."/>
        <xsd:field xpath="@DefaultStartActivitySetId"/>
    </xsd:keyref>
    <!-- check that specified default start activity exists (note: incomplete test, does not constrain to optional activtyset specified by
DefaultStartActivitySetId) -->
    <xsd:keyref name="DefaultStartActivityIdRef.WorkflowProcess" refer="xpd:ActivityIds.WorkflowProcess">
        <xsd:selector xpath="."/>
        <xsd:field xpath="@DefaultStartActivityId"/>
    </xsd:keyref>
    <!-- check that the activityset specified in a blockactivity exists -->
    <xsd:keyref name="BlockActivityActivitySetIdRef.WorkflowProcess" refer="xpd:ActivitySetIds.WorkflowProcess">
        <xsd:selector xpath="//xpd:BlockActivity"/>
        <xsd:field xpath="@ActivitySetId"/>
    </xsd:keyref>
    <!-- check that the start activity specified in a blockactivity exists (note: incomplete test, does not constrain to activtyset
specified by ActivitySetId) -->
    <xsd:keyref name="BlockActivityStartActivityIdRef.WorkflowProcess" refer="xpd:ActivityIds.WorkflowProcess">
        <xsd:selector xpath="//xpd:BlockActivity"/>
        <xsd:field xpath="@StartActivityId"/>
    </xsd:keyref>
    <!-- check that the from and to specified in a transition exists -->
    <xsd:keyref name="TransitionFromRef.WorkflowProcess" refer="xpd:ActivityIdsTopLevel.WorkflowProcess">
        <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
        <xsd:field xpath="@From"/>
    </xsd:keyref>
    <xsd:keyref name="TransitionToRef.WorkflowProcess" refer="xpd:ActivityIdsTopLevel.WorkflowProcess">
        <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
        <xsd:field xpath="@To"/>
    </xsd:keyref>
    <!-- check that the id specified in a transitionref exists -->
    <xsd:keyref name="TransitionRefIdRef.WorkflowProcess" refer="xpd:TransitionIdsTopLevel.WorkflowProcess">
        <xsd:selector
xpath="/xpd:Activities/xpd:Activity/xpd:TransitionRestrictions/xpd:TransitionRestriction/xpd:Split/xpd:TransitionRefs/xpd:Transiti
onRef"/>
        <xsd:field xpath="@Id"/>
    </xsd:keyref>
</xsd:element>

```

	Description
AccessLevel	The Access level of a process may be either PUBLIC or PRIVATE. If PUBLIC the process may be invoked by an external system or application. A process with private access may only be invoked from a SubFlow/subprocess Activity (see section 7.6.5.3.10). Use is optional and default is PUBLIC.
Activities	A list of activities that comprise the process. See section 7.6.
ActivitySets	A list of self contained sets of activities and transitions. Used to represent a BPMN EmbeddedSubprocess and EventSubprocess.
AdHoc	See section 7.5.9.

	Description
AdHocOrdering	See section 7.5.9.
AdHocCompletionCondition	See section 7.5.9.
Applications	A list of Application Declarations. See section 7.3.
Assignments	A list of data field <b>assignments</b> . See section 7.1.6.
DataAssociations	A list of DataAssociation elements defined within the process.
DataFields	A list of Relevant data fields defined for the process. See section 7.12.
DataObjects	A list of DataObject elements defined within the process. See section 7.1.10.2
DefaultStartActivityId	If present, DefaultStartActivityId must be the id of a start activity <ul style="list-style-type: none"> <li>• In the Default StartActivitySet if that's present</li> <li>• In the top level process activities otherwise</li> </ul>
DefaultStartActivitySetId	If present, DefaultStartActivitySetId must be the id of an Activity Set in the process.
EnableInstanceCompensation	See section 7.5.8.
ExtendedAttributes	Optional vendor-defined extensions to meet implementation needs. See section 7.1.1.
FormalParameters	A list of parameters that may be passed to the process. See section 7.1.4.
Id	Used to identify the process.
InputSets (BPMN alternative to Formal Parameters)	The InputSets attribute defines the data requirements for input to the Process. Zero or more InputSets MAY be defined. Each Input set is sufficient to allow the Process to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow). See section 7.6.10.
Name	Text Used to identify the process.
Object	See section 7.1.9.4.
OutputSets (BPMN alternative to Formal Parameters)	The OutputSets attribute defines the data requirements for output from the Process. Zero or more OutputSets MAY be defined. At the completion of the Process, only one of the OutputSets may be produced. It is up to the implementation of the Process to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the Process. See section 7.6.11.
Participants	A list of resources used in implementing the process. See section 7.11.
PartnerLinks	Partner links used by this process. See section 7.8.2.
ProcessHeader	A set of elements specifying process characteristics.
ProcessType	BPMN types: None, Private, Abstract, Collaboration. See section 7.5.5.
RedefinableHeader	A set of elements and attributes used by both the Package and Process definitions.
Status	See section 7.5.6.
SuppressJoinFailure	See section 7.5.7.
Transitions	A list of the transitions that connect the process activities. See section 7.7.

Table : Process Definition

### 7.5.2. Process Definition Header

The process definition header keeps all information specific for a process definition such as process version, priority, duration of validity, etc.

	Description
--	-------------

	Description
Created	Creation date of process definition.
Description	Short textual description of the process.
Duration	Expected duration time to perform a task in units of DurationUnit.
DurationUnit	Describes the default unit to be applied to an integer duration value that has no unit tag. Possible units are: Y - year M - month D - day H - hour m - minute s – second
Limit	Expected duration for time management purposes (e.g. starting an escalation procedure etc.) in units of DurationUnit. It is counted from the starting date/time of the Process. The consequences of reaching the limit value are not defined in this document (i.e. vendor specific).
Priority	The priority of the process type. The units are defined in the Package header priority units.
TimeEstimation	Grouping of waiting time, working time, and duration. Used for simulation purposes.
ValidFrom	The date that the process definition is active from. Empty string means system date. Default: Inherited from Model Definition.
ValidTo	The date at which the process definition becomes valid. Empty string means unlimited validity. Default: Inherited from Model Definition.
WaitingTime	Describes the amount of time, which is needed to prepare the performance of the task (time estimation) (waiting time is provided by the analysis environment and may be updated by the runtime environment) in units of DurationUnit.
WorkingTime	Describes the amount of time the performer of the activity needs to perform the task (time estimation) (working time is needed for analysis purposes and is provided by the evaluation of runtime parameters) in units of DurationUnit.

*Table : Process Definition Header***7.5.3. Process Redefinable Header**

Refer to Redefinable Header at the Package level: 7.2.2 .

	Description
Author	Name of the author of this process definition. (The one who put it into the repository).
Codepage	The codepage used for the text parts. Default: Inherited from Model Definition.
Countrykey	Country code based on ISO 3166. It could be either the three digits country code number, or the two alpha characters country codes. Default: Inherited from Model Definition.

	Description
PublicationStatus	Status of the Process Definition. Default: Inherited from Model Definition.  UNDER_REVISION  RELEASED  UNDER_TEST
Responsible(s)	Participant, who is responsible for this process (usually an Organisational Unit or a Human). It is assumed that the responsible is the supervisor during execution of the process. Default: Inherited from Model Definition.
Version	Version of this process definition.

*Table : Process Redefinable Header*

### 7.5.4. Activity Set/Embedded SubProcess

An activity set is a self-contained set of activities and transitions. Transitions in the set should refer only to activities in the same set and there should be no transitions into or out of the set. Activity sets can be executed by block activities (see section 7.6.3). In the case of Event Sub-Process the execution of the Activity set is triggered by an Event (see section 7.5.4.1). An Activity Set is re-usable; it may be referred to by more than one Block Activity.

The xml at the end of ActivitySet, starting with keyname, ensures referential integrity.

```

<xsd:key name="ActivityIds.ActivitySet">
  <xsd:selector xpath="/xpd:Activities/xpd:Activity"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<xsd:key name="TransitionIds.ActivitySet">
  <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
  <xsd:field xpath="@Id"/>
</xsd:key>
<!-- check that the default start activity id exists -->
<xsd:keyref name="DefaultStartActivityIdRef.ActivitySet" refer="xpd:ActivityIds.ActivitySet">
  <xsd:selector xpath="."/>
  <xsd:field xpath="@DefaultStartActivityId"/>
</xsd:keyref>
<!-- check that the from and to specified in a transition exists -->
<xsd:keyref name="TransitionFromRef.ActivitySet" refer="xpd:ActivityIds.ActivitySet">
  <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
  <xsd:field xpath="@From"/>
</xsd:keyref>
<xsd:keyref name="TransitionToRef.ActivitySet" refer="xpd:ActivityIds.ActivitySet">
  <xsd:selector xpath="/xpd:Transitions/xpd:Transition"/>
  <xsd:field xpath="@To"/>
</xsd:keyref>
<!-- check that the id specified in a transitionref exists -->
<xsd:keyref name="TransitionRefIdRef.ActivitySet" refer="xpd:TransitionIds.ActivitySet">
  <xsd:selector
xpath="/xpd:Activities/xpd:Activity/xpd:TransitionRestrictions/xpd:TransitionRestriction/xpd:Split/xpd:TransitionRefs/xpd:Transiti
onRef"/>
  <xsd:field xpath="@Id"/>
</xsd:keyref>
</xsd:element>

```

	Description
Activities	A list of activities that comprise the process. See section 7.6.
AdHoc	See section 7.5.9.
AdHocOrdering	See section 7.5.9.
AdHocCompletionCondi tion	See section 7.5.9.

	Description
DataAssociations.	To support BPMN2.0.
DataObjects	To Support BPMN2.0. See section 7.1.10.2
DefaultStartActivityId	Unless otherwise specified in the ActivitySet invocation (BlockActivity 7.6.3), this is where the activity set will start executing. If present, it must be the id of a start activity in the activity set. If not present it is assumed the activity set contains exactly one start activity.
Id	Used to identify the process.
Name	Name of the activity set/embedded sub-process.
Object	See section 7.1.9.4.
Transitions	A list of the transitions that connect the process activities. See section 7.7.
Triggered by Event	If true indicates that it is a BPMN2.0 Event Sub-Process.

Table : ActivitySet

#### 7.5.4.1. Event Sub-Process

An Event Sub-Process is defined in much the same way as an Embedded Sub-Process, except that the TriggeredByEvent attribute is set to True. An Event Sub-Process is not instantiated by a BlockActivity, but rather by the occurrence of an Event.

An **Event Sub-Process** is a specialized **Sub-Process** that is used within a **Process** (or **Sub-Process**).

An **Event Sub-Process** is not part of the *normal flow* of its parent **Process**—there are no *incoming* or *outgoing Sequence Flow*.

- An **Event Sub-Process** MUST NOT have any *incoming* or *outgoing Sequence Flow*.

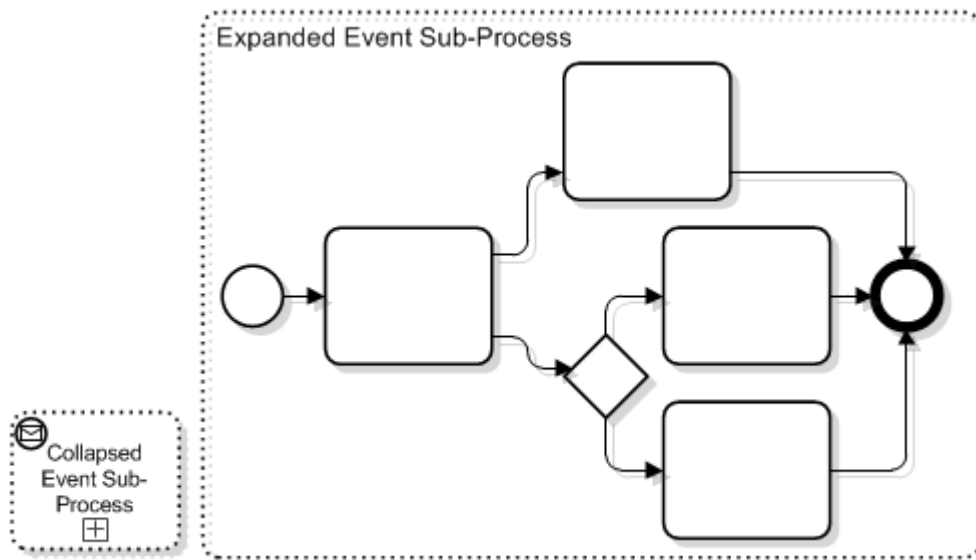
An **Event Sub-Process** may or may not occur while the parent **Process** is active, but it is possible that it will occur many times. Unlike a standard **Sub-Process**, which uses the flow of the parent **Process** as a *trigger*, an **Event Sub-Process** has a **Start Event** with a *trigger*. Each time the **Start Event** is triggered while the parent **Process** is active, then the **Event Sub-Process** will start.

- The **Start Event** of an **Event Sub-Process** MUST have a defined *trigger*.
  - The **Start Event** MUST be from the following types: Message, Error, Escalation, Compensation, Conditional, Signal, and Multiple
  - An **Event Sub-Process** MUST have one and only one **Start Event**.

An **Event Sub-Process** object shares the same basic shape as the **Sub-Process** object, which is a rounded rectangle.

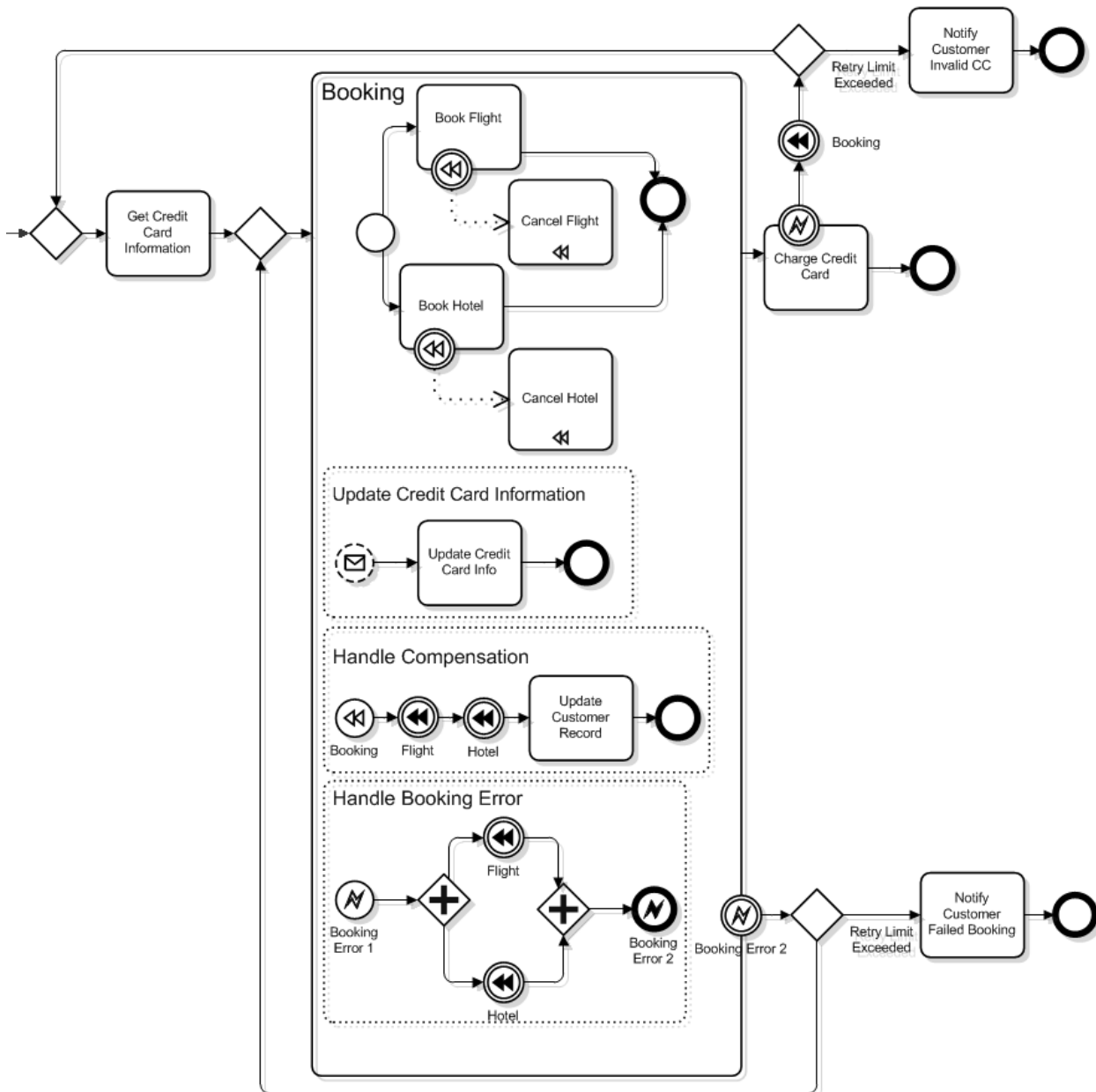
- An **Event Sub-Process** is a rounded corner rectangle that MUST be drawn with a single thin dotted line.
  - If the **Event Sub-Process** is collapsed, then its **Start Event** will be used as a marker in the upper left corner of the shape.





There are two (2) possible consequences to the parent **Process** when an **Event Sub-Process** is triggered: 1) the parent **Process** can be interrupted, and 2) the parent **Process** can continue its work (not interrupted). This is determined by the type of **Start Event** that is used.

The following figure provides an example of a **Sub-Process** that includes three (3) **Event Sub-Processes**. The first **Event Sub-Process** is triggered by a **Message**, does not interrupt the **Sub-Process**, and can occur multiple times. The second **Event Sub-Process** is used for *compensation* and will only occur after the **Sub-Process** has completed. The third **Event Sub-Process** handles errors that occur while the **Sub-Process** is active and will stop (interrupt) the **Sub- Process** if triggered.



### 7.5.5. ProcessType in BPMN mapping to WS-BPEL

ProcessType is an attribute that provides information about which lower level language the Pool will be mapped to. By default, the ProcessType is None (or undefined). A Private ProcessType MAY be mapped to an executable WS-BPEL process. An Abstract ProcessType is also called the public interface of a process (or other web services) and MAY be mapped to an abstract WS-BPEL process. A Collaboration ProcessType will have two Lanes that represent business roles (e.g., buyer or seller) and will show the interactions between these roles. These Pools MAY be mapped to languages such as ebXML or WS Choreography. If the Process is to be used to create a WS-BPEL document, then the attribute MUST be set to Private or Abstract.

### 7.5.6. Status

The Status of a Process is determined when the Process is being executed by a process engine. The Status of a Process can be used within Assignment Expressions.

The following states are recognized:

- None
- Ready
- Active
- Cancelled
- Aborting
- Aborted
- Completing
- Completed

### **7.5.7. SuppressJoinFailure**

This attribute is included for mapping to WS-BPEL. This specifies whether or not a WS-BPEL joinFailure fault will be suppressed for all activities in the WS-BPEL process.

### **7.5.8. EnableInstanceCompensation**

This attribute is included for mapping to WS-BPEL. It specifies whether or not a compensation can be performed after the Process has completed normally.

### **7.5.9. AdHoc**

AdHoc is a boolean attribute, which has a default of False. This specifies whether the Process is Ad Hoc or not. The activities within an Ad Hoc Process are not controlled or sequenced in a particular order; their performance is determined by the performers of the activities.

If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocOrdering attribute MUST be included. This attribute defines if the activities within the Process can be performed in Parallel or must be performed sequentially. The default setting is Parallel and the setting of Sequential is a restriction on the performance that may be required due to shared resources.

If the Process is Ad Hoc (the AdHoc attribute is True), then the AdHocCompletionCondition attribute MUST be included. This attribute defines the conditions when the Process will end.

## **7.6. Process Activity**

The Activity Definition is used to define each elementary activity that makes up a process. Attributes may be defined to specify activity control information, implementation alternatives, performer assignment, runtime relevant information like priority, and data used specifically in BPR and simulation situations (and not used within enactment). In addition, restrictions on data access and to transition evaluation (e.g. Split and Join) can be described. Mandatory attributes are used to define the activity identifier and type; a small number of other attributes are optional but have common usage across all activity types. Other attribute usage depends upon the activity type as shown in the table below.

For the Activity identifier and name the scope is the surrounding process.

The activity description is used to describe several different activity types. All these activities share the same (common) general activity attributes, but the usage of other attributes, particularly participant and application assignment and the use of relevant data field may be specialized to the activity type. The following table identifies the usage of other attributes / entity types for the different activity types.

Entity Types (usage within Activity Type)	Activity Type				
	Implementation Type			Route/GateWay/Event	BlockActivity
	None	Application/ Task	SubFlow/subprocess		
Transition Restriction	Normal	Normal	Normal, plus subflow call / return within activity	Normal; any additional controls implemented within Route activity	Normal; refers to activities within same context, not to activities within ActivitySet
Performers/Participant Assignment	Normal	Normal/See Task details	N/A	N/A	N/A
Application Assignment	None	Yes/See task details	N/A	N/A	N/A
Use of relevant data field	Normal	Normal	May be used in parameter passing	May be used in routing control conditions	May be used in routing control conditions

Table : Entity type relationships for different Activity types

BPMN provides specific graphics for the activity types supported.

In BPMN 1.x:



In BPMN 2.0:



Figure 7.: BPMN Activity Types

In XPDL a Reusable SubProcess is called a **Subflow** and Embedded SubProcess is called a **BlockActivity**. GateWay is called a **RouteActivity**. Event is a new construct for XPDL 2.0. Note that BPMN does not use Application; this is terminology from XPDL. BPMN2.0 introduces a new type of SubProcess: EventSubProcess. Its definition in XPDL is via ActivitySets (i.e. similar to EmbeddedSubProcess) but its instantiation is not by a BlockActivity but rather by a BPMN Event. See details in section 7.6.4 on Events. A collapsed Event SubProcess is depicted as follows:



Where its **Start Event** will be used as a marker in the upper left corner of the shape. (In this example a Message Event).

## Notes on usage:

Transition restrictions, subflow, and route activities are described in the sequel. In general, normal transition restrictions may be declared at the level of the activity boundary within the surrounding process, whereas specialized flow conditions (subflow, or the internal part of a route activity) operate “internal” to the activity (but may reference activities within the surrounding process definition). The following diagram illustrates the generic structure of an activity and the above variants.

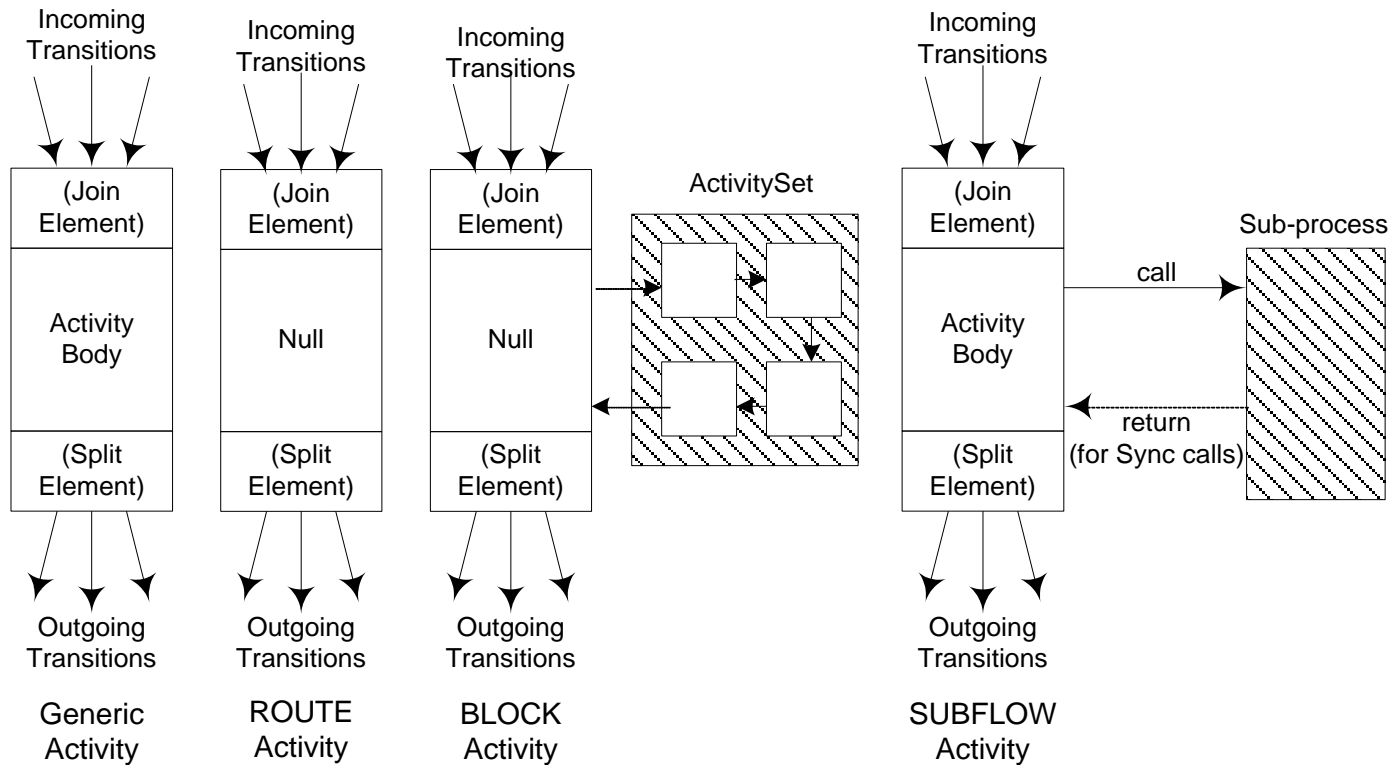


Figure 7.: Activity Structures & Transition Conditions

Note that any activity that has multiple outgoing transitions (sequence flow) must have a Split Element containing the list of outgoing transitions. (See sections 7.6.9 and 7.6.9.2). This also applies to the BPMN task, subprocess and gateway constructs.

Where the implementation type is NONE, the activity is manually controlled and its completion must be explicitly signaled to the process or management system. Such activities might typically comprise instructions to the participant to undertake a non-automated task of some type and inform a supervisor when completed.

Relevant data field may (potentially) be referenced within any activity although its use in manual activities is undefined through the process definition. Where an activity is of type SubFlow any in-parameters passed to the called (sub-) process must have been declared as relevant data fields within the calling process / activity definition, or have been inherited from the surrounding package. (Similar requirements apply to any out-parameters returned to the calling process.) Routing and block activities may refer to such data within conditional expressions within the join/split control logic.

	Description
Assignments	A list of data field assignments. See section 7.1.6.
BlockActivity	An Activity that executes an ActivitySet. See 7.6.3.
CompletionQuantity	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must be generated from the activity. This number of Tokens will be sent down any outgoing Sequence Flow (assuming any Sequence Flow Conditions are satisfied).

	Description
Deadline	Specification of a deadline and action to be taken if it is reached. It is better to use the BPMN timer event to provide this functionality.
Description	Textual description of the activity.
Documentation	The address (e.g. path- and filename) for a help file or a description file of the activity.
DataFields(Properties)	Allows declaration of relevant data local to the activity. See section 7.12 .
Event	See section 7.6.4.
ExtendedAttributes	Optional extensions to meet individual implementation needs.
FinishMode	Describes how the system operates at the end of the Activity.
Icon	Alternative graphics for an icon to represent the activity in a graphical modeller. May be used to override the modeller icon for an activity. This may be deprecated in the future.
Id	Used to identify the process activity.
Implementation	A "regular" Activity. Mandatory if not a Route. Alternative implementations are "no", "Task", "SubFlow" or "Reference".
InputSets	See section 7.6.10.
IORules	The IORules attribute is a collection of expressions, each of which specifies the required relationship between one input and one output. That is, if the activity is instantiated with a specified input, that activity shall complete with the specified output.
IsATransaction	If the activity is a block activity or is implemented as a subflow IsATransaction determines whether or not the behavior of the Sub-Process will be treated as a Transaction.
IsForCompensation	Default value: false.
Limit	Expected duration for time management purposes (e.g. starting an escalation procedure etc.) in units of DurationUnit. It is counted from the starting date/time of the Process. The consequences of reaching the limit value are not defined in this document (i.e. vendor specific). Note that BPMN provides Timer Events which can be attached to the boundary of a regular or subflow/subprocess activity.
Loop	See section 7.6.13.
Name	Text Used to identify the process activity.
NodeGraphicsInfos	Optional. See section 7.1.1.
Object	See section 7.1.9.4.
OutputSets	See section 7.6.11.
Performers	List of Links to entity participants. Each Performer may be an expression. Default: Any Participant.
Priority	A value that describes the initial priority of this activity when it starts execution. If this attribute is not defined but a priority is defined in the Process definition then that is used. By default it is assumed that the priority levels are the natural numbers starting with zero, and that the higher the value the higher the priority (i.e.: 0, 1,..., n).
Route	A "dummy" Activity used for routing. A BPMN Gateway.
SimulationInformation	Estimations for simulation of an Activity. No default. See section 7.6.8.
StartActivity	Designates the first activity to be executed when the process is instantiated. Used when there is no other way to determine this. Conflicts with BPMN StartEvent and no process definition should contain both.
StartMode	Describes how the execution of an Activity is triggered.
StartQuantity	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that must arrive before the activity can begin.
Status	Status values are assigned during execution. Status can be treated as a property and used in expressions local to an Activity.

	Description
TransactionRef Editors note: are transactions reusable by multiple activities? Must be resolved in BPMN.	If the IsATransaction attribute is False, then a Transaction MUST NOT be identified. If the IsATransaction attribute is True, then a Transaction MUST be identified. Note that Transactions that are in different Pools and are connected through Message Flow MUST have the same TransactionId. See section 7.6.12.
TransitionRestrictions	Provides further restrictions and context-related semantics description of Transitions. All activities (including Gateways) with multiple outgoing transitions (sequence flow) must have a TransitionRestrictions Split element with a list of references to the outgoing transition elements. (See 7.6.9.2).

Table : Process Activity

### 7.6.1. Execution Control Attributes

These are attributes of an Activity that allow the definition of various activity-specific features for Activity execution control. Refer to the Table for Process Activity.

Automation mode defines the degree of automation when triggering and terminating an activity. There are two automation modes:

- **Automatic mode** is fully controlled by the process or workflow engine, i.e. the engine proceeds with execution of the activity within the process automatically, as soon as any incoming transition conditions are satisfied. Similarly, completion of the activity and progression to any post activity conditional logic occurs automatically on termination of the final invoked application.
- **Manual mode** requires explicit user interaction to cause activity start or finish. In such systems the activity start and/or completion is as a result of explicit user action.

The automation modes can be specified independently for the *start* and *end* of an Activity.

### 7.6.2. Route Activity

The Route Activity is a “dummy” Activity that permits the explicit expression of the behavior of diverging (split) or converging (join) sequence flow in a process activity. A Route Activity is a Gateway activity. (In BPMN, the split and join behavior for non-gateway activities is always the same: joins are always Exclusive and splits are always Inclusive).

The Route Activity also permits the expression of "cascading" Transition conditions between activities (e.g. of the type "IF condition-1 THEN TO Activity-1 ELSE IF condition-2 THEN TO Activity-2 ELSE Activity-3 ENDIF"). Some vendors might implement "cascading" transition conditions directly without requiring an activity counterpart for a route; others might require it. Wherever possible vendors and process designers are encouraged to structure such cascading conditions using an Exclusive Gateway or an XOR split from the outgoing activity. Certain transition combinations cannot be expressed within a single transition list from the outgoing activity or a single incoming list to an activity. These cases require the use of one or more dummy activities; examples are:

- Combination of XOR and AND split conditions on outgoing transitions from an activity.
- Combination of XOR and AND join conditions on incoming transitions to an activity.
- Transitions involving conditional AND joins of a subset of threads, with continuation of individual threads.

The TransitionRestrictions (sub elements of Activity) provide the “order of evaluation” for multiple outgoing transitions (which is especially significant for XOR splits). **Any activity (including route activities) with multiple outgoing transitions (sequence flow) must have a SPLIT transition restriction with a list of references to the outgoing transitions (see sections 7.6.9 and 7.6.9.2).** If the Type and ExclusiveType attributes of the TransitionRestriction/Split | Join element are also used then they MUST match the type specified in the GatewayType and ExclusiveType attributes

of the Route element.

A route activity has neither a performer nor an application.

For simulation purposes the following simulation data values should be assumed: Duration "0", Cost "0", WorkingTime "0", WaitingTime "0". For Priority and Instantiation the maximum value should be assumed.

	Description
GatewayType	Specifies the split / join behavior of this gateway activity.  Note that the enumerations XOR, OR and AND are deprecated and replaced by the new enumerations Exclusive, Inclusive and Parallel respectively. It is recommended that modellers are capable of reading deprecated values but always write the new enumerations.  When used for BPMN gateways, the GatewayType MUST be specified.  Values: XOR   OR   AND   Exclusive   Inclusive   Parallel   Complex
ExclusiveType	For BPMN "Exclusive" gateways this attribute MAY be specified when the GatewayType is specified as "Exclusive", this attribute is used to specify whether the gateway is an Exclusive <b>Data</b> -Based gateway (default) or an Exclusive <b>Event</b> -Based gateway.  For other gateway types, this attribute MUST NOT be specified.  Values: Data   Event
XORType	<b>Deprecated</b> (in favour of ExclusiveType). It is recommended that modellers are capable of reading from this attribute (in the absence of ExclusiveType) but always use the new ExclusiveType when writing.
Instantiate	Exclusive Event-Based Gateways can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Gateway is the first element after the Start Event or a starting Gateway if there is no Start Event (i.e., there are no incoming Sequence Flow).
ParallelEventBased	Used for BPMN 2.0. GatewayType must be set to Parallel and Instantiate to true.
MarkerVisible	This attribute determines if the XOR Marker is displayed in the center of the Gateway diamond (an "X"). The marker is displayed if the attribute is True and it is not displayed if the attribute is False. By default, the marker is not displayed.
IncomingCondition	For a Complex Gateway, if there are multiple incoming Sequence Flow, an IncomingCondition expression MUST be set by the modeler. This will consist of an expression that can reference Sequence Flow names and/or Process Properties (Data).
OutgoingCondition	For a Complex Gateway, if there are multiple outgoing Sequence Flow, an OutgoingCondition expression MUST be set by the modeler. This will consist of an expression that can reference (outgoing) Sequence Flow Ids and/or Process Properties (Data).

Table : Route Activity

#### 7.6.2.1. Gateway Activity

Gateways are modeling elements that are used to control how Sequence Flows interact as they converge and diverge within a Process. If the flow does not need to be controlled, then a Gateway is not needed. The term "Gateway" implies that there is a gating mechanism that either allows or disallows passage through the Gateway--that is, as Tokens arrive at a Gateway, they can be merged together on input and/or split apart on output as the Gateway mechanisms are invoked. To be more descriptive, a Gateway is actually a collection of "Gates." Although the Gates are not graphically depicted, the Gates are used by the Sequence Flow to connect to or from the Gateway.

BPMN Gateway activities are represented by Route Activities.

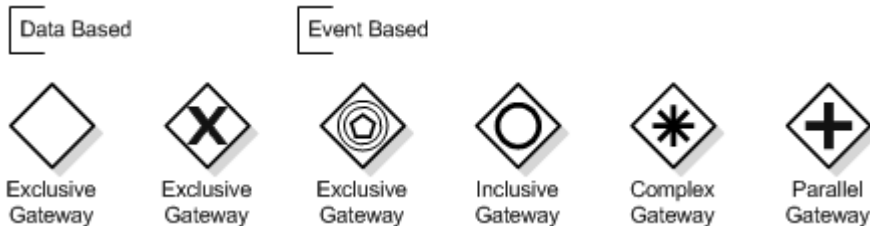
The TransitionRestrictions (sub elements of Activity) provide the "order of evaluation" for multiple outgoing transitions (which is especially significant for XOR splits). **Any activity (including route activities) with multiple outgoing transitions (sequence flow) must have a SPLIT transition restriction with a list of references to the outgoing transitions (see sections 7.6.9 and 7.6.9.2).** If the Type and ExclusiveType attributes of the TransitionRestriction/Split | Join element is also used then they MUST match the corresponding GatewayType and ExclusiveType attributes specified in the Route element.

A Gateway is used to control the divergence and convergence of Sequence Flow. Thus, it will determine branching,



forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control. Exclusive gateways can be either “data” or “event” based.

BPMN provides specific graphics for the gateway types supported.



BPMN 2.0 added two more shapes which are variations on the **Event Based GateWay** to start a process:

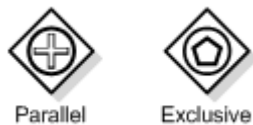


Figure 7.: Gateway Types

**Note** – Although the shape of a Gateway is a diamond, it is not a requirement that incoming and outgoing Sequence Flow must connect to the corners of the diamond. Sequence Flow can connect to any position on the boundary of the Gateway shape.

- The internal marker associated with the Gateway **MUST** be placed inside the shape, in any size or location, depending on the preference of the modeler or modeling tool vendor, with the exception that the marker for the Data- Based Exclusive Gateway is not required.
- The ‘X’ internal marker is optional in the data based exclusive gateway graphic. A process diagram should be consistent in its use of this internal marker; modelers should avoid using both variants in a process diagram as that may confuse readers.

The Gateways will control the flow of both diverging and/or converging Sequence Flow. That is, a particular Gateway could have multiple input Gates and multiple output Gates at the same time (there is one Sequence Flow per Gate). The type of Gateway will determine the same type of behavior for both the diverging and converging Sequence Flow. Modelers and modeling tools may want to enforce a best practice of a Gateway only performing one of these functions. Thus, it would take two sequential Gateways to first converge and then diverge the Sequence Flow.

#### 7.6.2.2. Examples of Gateways and their Representation

##### 7.6.2.2.1. XOR Gate – Data based

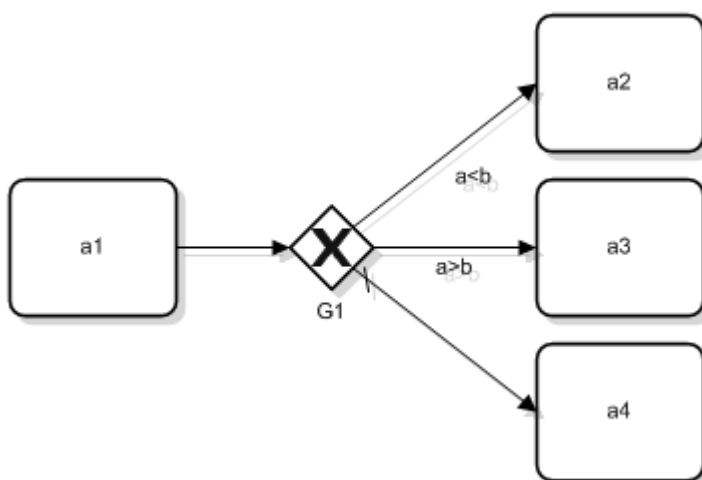


Figure 7.: Exclusive Decision – Data Based

```

<Activities>
  <Activity Id="3" Name="a1"/>
  <Activity Id="4" Name="g1">
    <Route GatewayType="Exclusive" MarkerVisible="TRUE"/>
    <TransitionRestrictions>
      <TransitionRestriction>
        <Split Type=" Exclusive ">
          <TransitionRefs>
            <TransitionRef Id="9"/>
            <TransitionRef Id="10"/>
            <TransitionRef Id="11"/>
          </TransitionRefs>
        </Split>
      </TransitionRestriction>
    </TransitionRestrictions>
  </Activity>
  <Activity Id="5" Name="a2"/>
  <Activity Id="6" Name="a3"/>
  <Activity Id="7" Name="a4"/>
</Activities>
<Transitions>
  <Transition Id="8" Name="" From="3" To="4" FlowType="SequenceFlow"/>
  <Transition Id="9" Name="" From="4" To="5" FlowType="SequenceFlow">
    <Condition Type="CONDITION">a<lt;b</Condition>
  </Transition>
  <Transition Id="10" Name="" From="4" To="6" FlowType="SequenceFlow">
    <Condition Type="CONDITION">a<lt;b</Condition>
  </Transition>
  <Transition Id="11" Name="" From="4" To="7" FlowType="SequenceFlow">
    <Condition Type="OTHERWISE"/>
  </Transition>
</Transitions>

```

#### 7.6.2.2.2. Merge

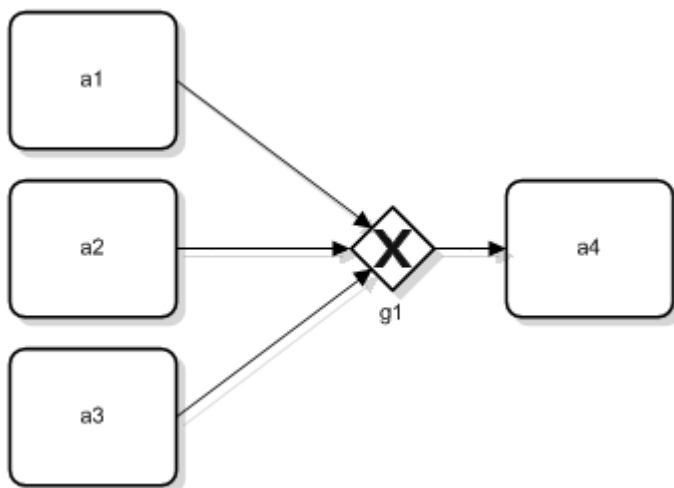


Figure 7.: Exclusive Merge

```

<Activities>
  <Activity Id="3" Name="a1"/>
  <Activity Id="4" Name="a2"/>
  <Activity Id="5" Name="a3"/>
  <Activity Id="6" Name="g1">
    <Route GatewayType="Exclusive" MarkerVisible="TRUE"/>
  </Activity>
  <Activity Id="7" Name="a4"/>
</Activities>
<Transitions>
  <Transition Id="8" Name="" From="3" To="6" FlowType="SequenceFlow"/>
  <Transition Id="9" Name="" From="4" To="6" FlowType="SequenceFlow"/>
  <Transition Id="10" Name="" From="5" To="6" FlowType="SequenceFlow"/>
  <Transition Id="11" Name="" From="6" To="7" FlowType="SequenceFlow"/>
</Transitions>

```

### 7.6.2.3. BPMN Semantics for Gateway Types

#### 7.6.2.3.1. Common Gateway Sequence Flow Connections

This section applies to all Gateways. Additional Sequence Flow Connection rules will be specified for each type of

Gateway in the sections below.

- A Gateway MAY be a target for Sequence Flow; it can have zero or more incoming Sequence Flows. An incoming Flow MAY be from an alternative path or a parallel path
  - If the Gateway does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Gateway's divergence behavior, depending on the GatewayType attribute (see below), SHALL be performed when the Process is instantiated.
- A Gateway MAY be a source of Sequence Flow; it can have zero or more outgoing Flows.
- A Gateway MAY have both multiple incoming and outgoing Sequence Flows.

**Note** – The incoming and outgoing Sequence Flows are not required to attach to the corners of the Gateway's diamond shape. Sequence Flow can attach to any location on the boundary of a Gateway.

#### 7.6.2.3.2. *Common Gateway Message Flow Connections*

This section applies to all Gateways.

- A Gateway MUST NOT be a target for Message Flow.
- A Gateway MUST NOT be a source for Message Flow.

#### 7.6.2.3.3. *Exclusive Gateways*

Exclusive Gateways (Decisions) are locations within a business process where the Sequence Flow can take two or more alternative paths. This is basically the “fork in the road” for a process. For a given performance (or instance) of the process, only one of the paths can be taken (this should not be confused with forking of paths). A Decision is not an activity from the business process perspective, but is a type of Gateway that controls the Sequence Flow between activities. It can be thought of as a question that is asked at that point in the Process. The question has a defined set of alternative answers (Gates). Each Decision Gate is associated with a condition expression found within an outgoing Sequence Flow. When a Gate is chosen during the performance of the Process, the corresponding Sequence Flow is then chosen. A Token arriving at the Decision would be directed down the appropriate path, based on the chosen Gate.

The Exclusive Decision has two or more outgoing Sequence Flows, but only one of them may be taken during the performance of the Process. Thus, the Exclusive Decision defines a set of alternative paths for the Token to take as it traverses the Flow. There are two types of Exclusive Decisions: Data-Based and Event-Based.

##### 7.6.2.3.3.1. *Data-Based*

The Data-Based Exclusive Gateways are the most commonly used type of Gateways. The set of Gates for Data-Based Exclusive Decisions is based on the boolean expression contained in the ConditionExpression attribute of the outgoing Sequence Flow of the Gateway. These expressions use the values of process data to determine which path should be taken (hence the name Data-Based).

- The Data-Based Exclusive Gateway MAY use a marker that is shaped like an “X” and is placed within the Gateway diamond to distinguish it from other Gateways. This marker is not required.
  - A Diagram SHOULD be consistent in the use of the “X” internal indicator. That is, a Diagram SHOULD NOT have some Gateways with an indicator and some Gateways without an indicator.

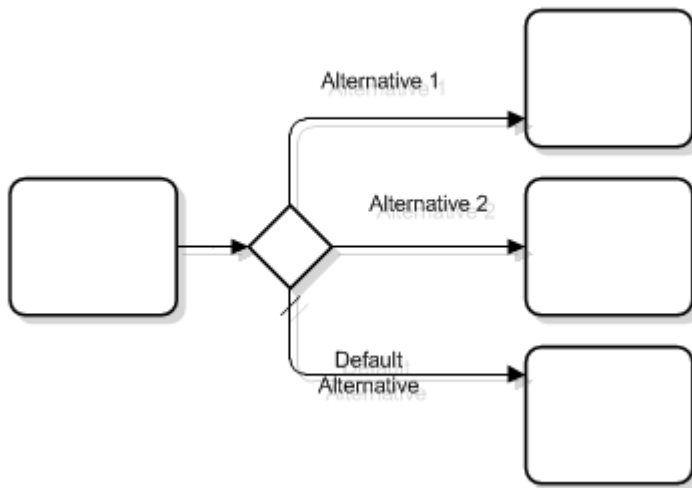


Figure 7.: Exclusive Decision without Indicator

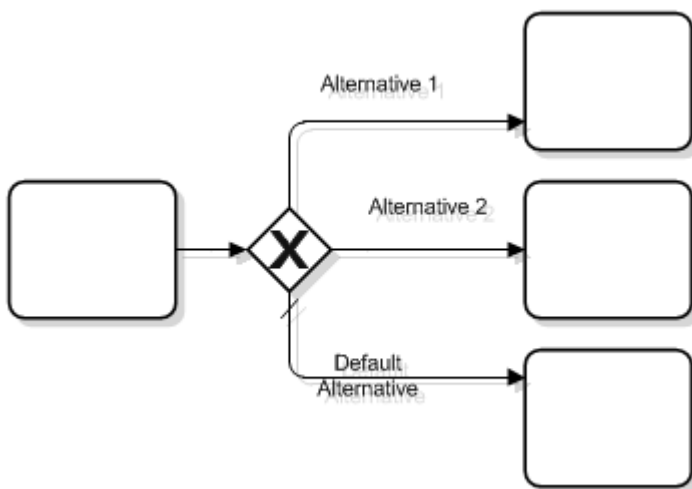
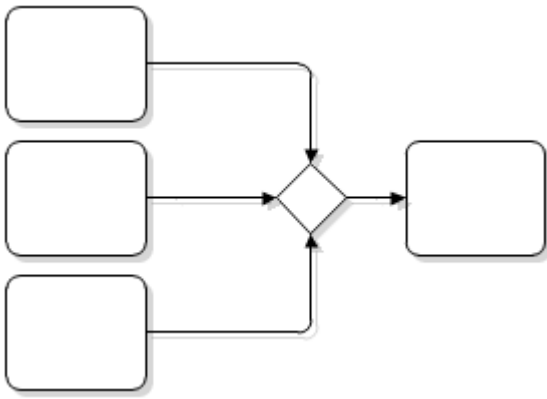


Figure 7.: Exclusive Decision with Indicator

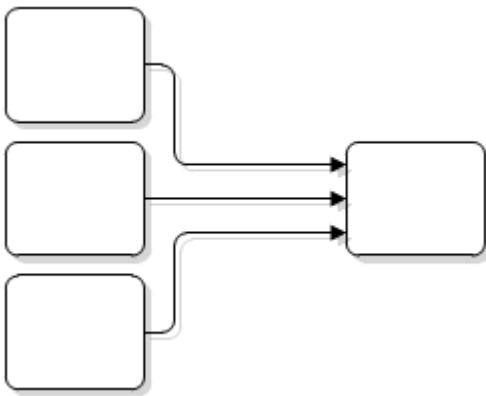
**Note** – The “X” internal indicator for the Data-Based Exclusive Gateway was included in BPMN to complete the set of indicators for the different types of Gateways. However, it is also understood that most modelers would be familiar with an empty decision diamond that represents an exclusive branching of the process and that most decisions would probably take this form. Thus, Data-Based Exclusive Gateway internal indicator was made optional so that modelers and modeling tools could create diagrams that would conform with the basic flow expectations of modelers.

The conditions for the alternative Gates should be evaluated in a specific order. (Note that the order is captured in the XPD L TransitionRestriction SPLIT element. See section 7.6.9.2). The first one that evaluates as TRUE will determine the Sequence Flow that will be taken. Since the behavior of this Gateway is exclusive, any other conditions that may actually be TRUE will be ignored--only one Gate can be chosen. One of the Gates may be “default” (or otherwise), and is the last Gate considered. This means that if none of the other Gates are chosen, then the default Gate will be chosen—along with its associated Sequence Flow. The default Gate is not mandatory for a Gateway. This means that if it is not used, then it is up to the modeler to insure that at least one Gate be valid at runtime. BPMN does not specify what will happen if there are no valid Gates. However, BPMN does specify that there MUST NOT be implicit flow and that all Normal Flow of a Process must be expressed through Sequence Flow. This would mean that a Process Model that has a Gateway that potentially does not have a valid Gate at runtime is an invalid model.



*Figure 7.: Exclusive Merge Without the Indicator*

Exclusive Gateways can also be used as a merge for alternative Sequence Flow, although it is rarely required for the modeler to use them this way. The merging behavior of the Gateway can also be modeled as seen below. The behavior of both are the same if all the incoming flow are alternative.



*Figure 7.: Uncontrolled Sequence Flow*

There are certain situations where an Exclusive Gateway is required to act as a merging object. In the figure below an Exclusive Gateway (labeled “Merge”) merges two alternative Sequence Flow that were generated by an upstream Decision. The alternative Sequence Flows are merged in preparation for a Parallel Gateway that synchronizes a set of parallel Sequence Flows that were generated even further upstream. If the merging Gateway was not used, then there would have been four incoming Sequence Flows into the Parallel Gateway. However, only three of the four Sequence Flows would ever pass a Token at one time. Thus, the Gateway would be waiting for a fourth Token that would never arrive. Thus, the Process would be stuck at the point of the Parallel Gateway.

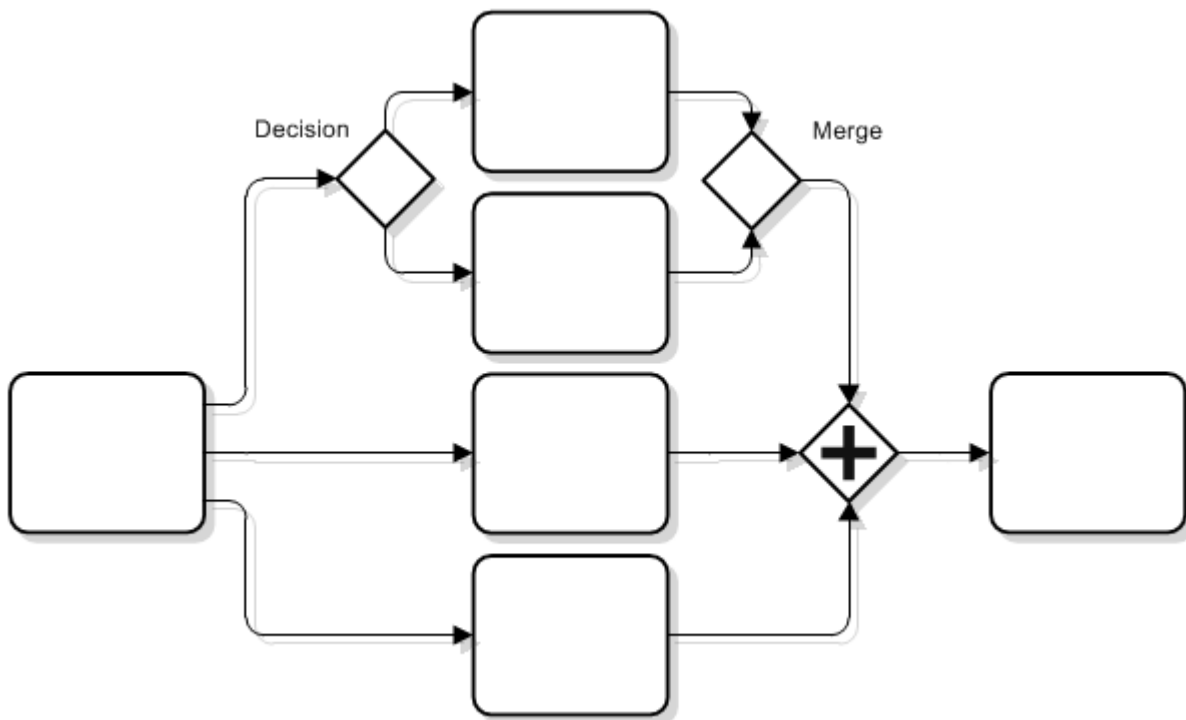


Figure 7.: Exclusive Gateway is required to act as a merging object

In simple situations, Exclusive Gateways need not be used for merging Sequence Flow, but there are more complex situations where they are required. Thus, a modeler should always be aware of the behavior of a situation where Sequence Flows are uncontrolled. Some modelers or modeling tools may, in fact, require that Exclusive Gateways be used in all situations as a matter of Best Practice.

#### Sequence Flow Connections for Data-Based Exclusive Gateway:

To define the exclusive nature of this Gateway's behavior for converging Sequence Flows:

- If there are multiple incoming Sequence Flows, all of them will be used to continue the flow of the Process (as if there were no Gateway). That is,
  - Process flow **SHALL** continue when a signal (a Token) arrives from any of a set of Sequence Flows.
    - Signals from other Sequence Flows within that set may arrive at other times and the flow will continue when they arrive as well, without consideration or synchronization of signals that have arrived from other Sequence Flows.

To define the exclusive nature of this Gateway's behavior for diverging Sequence Flows:

- If there are multiple outgoing Sequence Flows, then only one Gate (or the DefaultGate) **SHALL** be selected during performance of the Process
  - The Gate **SHALL** be chosen based on the result of evaluating the ConditionExpression that is defined for the Sequence Flow associated with the Gate
    - The Conditions associated with the Gates **SHALL** be evaluated in the order in which the Gates appear on the list for the Gateway.
    - If a ConditionExpression is evaluated as "TRUE," then that Gate **SHALL** be chosen and any Gates remaining on the list **MUST NOT** be evaluated.
    - If none of the ConditionExpressions for the Gates are evaluated as "TRUE," then the DefaultGate **SHALL** be chosen.

**Note** – If the Gateway does not have a DefaultGate and none of the Gate ConditionExpressions are evaluated as "TRUE," then the Process is considered to have an invalid model.

### 7.6.2.3.3.2. Event-Based

The inclusion of Event-Based Exclusive Gateways is the result of recent developments in the handling of distributed systems and was derived from the BPEL4WS pick. On the input side, their behavior is the same as a Data-Based Exclusive Gateway. On the output side, the basic idea is that this Decision represents a branching point in the process where the alternatives are based on events that occur at that point in the Process, rather than the evaluation of expressions using process data. A specific event, usually the receipt of a message, determines which of the paths will be taken. For example, if a company is waiting for a response from a customer, they will perform one set of activities if the customer responds “Yes” and another set of activities if the customer responds “No.” The customer’s response determines which path is taken. The identity of the Message determines which path is taken. That is, the “Yes” Message and the “No” message are different messages—they are not the same message with different values within a property of the Message. The receipt of the message can be modeled with a Task of TaskType Receive or an Intermediate Event with a Message Trigger. In addition to Messages, other Triggers for Intermediate Events can be used, such as Timers.

- The Event-Based Exclusive Gateway MUST use a marker that is the same as the Multiple Intermediate Event and is placed within the Gateway diamond to distinguish it from other Gateways.
- The Event-Based Exclusive Decisions are configured by having outgoing Sequence Flow target a Task of TaskType Receive or an Intermediate Event.
  - All of the outgoing Sequence Flow must have this type of target; there cannot be a mixing of condition expressions and Intermediate Events for a given Decision.

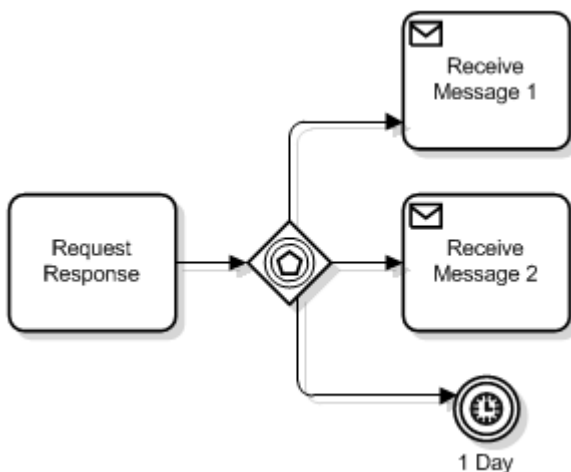


Figure 7.: An Event-Based Exclusive Gateway using Receive Tasks

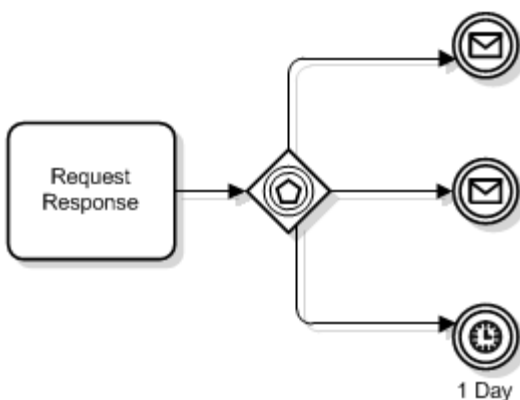


Figure 7.: An Event-Based Exclusive Gateway using Message Events

Because this Gateway is an Exclusive Gateway, the merging functionality for the Event-Based Exclusive Gateway is the same as the Data-Based Exclusive Gateway described in the previous section.

A Gateway can be used to start a Process: the receipt of any of the messages defined by the Gateway's configuration will instantiate the Process. Thus, the Gateway provides a set of alternative ways for a Process to begin. In order for the Gateway to Instantiate the Process it must meet one of the following conditions:

- The Process does not have a Start Event and the Gateway has no incoming Sequence Flow.
- The Incoming Sequence Flow for the Gateway has a source of a Start Event.
  - Note that no other incoming Sequence Flow is allowed for the Gateway (in particular, a loop connection from a downstream object).
- The Targets for the Gateway's outgoing Sequence Flow MUST NOT be a Timer Intermediate Event.
- The marker of the Gateway must be set to look like a Multiple Start Event (see Section 7.6.9.3.10).

In some situations a process may be instantiated by one of a set of messages while still requiring all of those messages for the execution of that process instance. To handle this there is a parallel variation of the Event Gateway. To instantiate a process using a parallel event-based Gateway:

- The marker of the Gateway must be set to look like a Parallel Multiple Start Event (see Section 7.6.9.3.11).
  - Note that for Event Gateways that do not instantiate the Process, a standard Parallel Gateway can be used to include parallel Events in the middle of a Process.

The Parallel Event Gateway is a type of race condition: the Process is instantiated when the first Event is triggered. However, the other Events of the Gateway configuration are not disabled but remain waiting and are expected to be triggered before the Process can (normally) complete. The Messages that trigger the Events of the Gateway configuration share the same correlation information.

#### **SequenceFlow Connections for Event-Based Exclusive Gateway**

To define the exclusive nature of this Gateway's behavior for converging Sequence Flows

- If there are multiple incoming Sequence Flows, all of them will be used to continue the flow of the Process (as if there were no Gateway). That is,
  - Process flow SHALL continue when a signal (a Token) arrives from any of a set of Sequence Flows.
    - Signals from other Sequence Flows within that set may arrive at other times and the flow will continue when they arrive as well, without consideration or synchronization of signals that have arrived from other Sequence Flows.

To define the exclusive nature of this Gateway's behavior for diverging Sequence Flows:

- Only one Gate SHALL be selected during performance of the Process.
  - The Gate SHALL be chosen based on the Target of the Gate's Sequence Flow.
    - If a Target is instantiated (e.g., a message is received or a time is exceeded), then that Gate SHALL be chosen and the remaining Gates MUST NOT be evaluated (i.e., their Targets will be disabled).
- The outgoing Sequence Flow Condition attribute MUST be set to None.
- The Target of the Gateway's outgoing Sequence Flow MUST be one of the following objects:
  - Task with the TaskType attribute set to Receive.
  - Intermediate Event with the Trigger attribute set to Message, Timer, Signal, or Conditional.
  - If one Gate Target is a Task, then an Intermediate Event with a Trigger Message MUST NOT be used as a Target for another Gate. That is, messages MUST be received by only Receive Tasks or only Message Events, but not a mixture of both for a given Gateway.

#### **7.6.2.3.4. Inclusive Gateways**

This Decision represents a branching point where Alternatives are based on conditional expressions contained within



outgoing Sequence Flow. However, in this case, the True evaluation of one condition expression does not exclude the evaluation of other condition expressions. All Sequence Flow with a True evaluation will be traversed by a Token. In some sense it is like a grouping of related independent Binary (Yes/No) Decisions--and can be modeled that way. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken.

**Note** – If none of the Inclusive Decision Gate ConditionExpressions are evaluated as “TRUE,” then the Process is considered to have an invalid model.

There are two mechanisms for modeling this type of Decision:

The first method for modeling Inclusive Decision situations does not actually use an Inclusive Gateway, but instead uses a collection of conditional Sequence Flows, marked with mini-diamonds--the Gates without the Gateway. Conditional Sequence Flow Transitions [XPDL term] have their ConditionType attribute set to CONDITION and the ConditionExpression attribute set to a boolean mathematical expression based on information available to the Process. These Sequence Flow are indicated by a “mini-diamond” marker at the start of the Sequence Flow line.

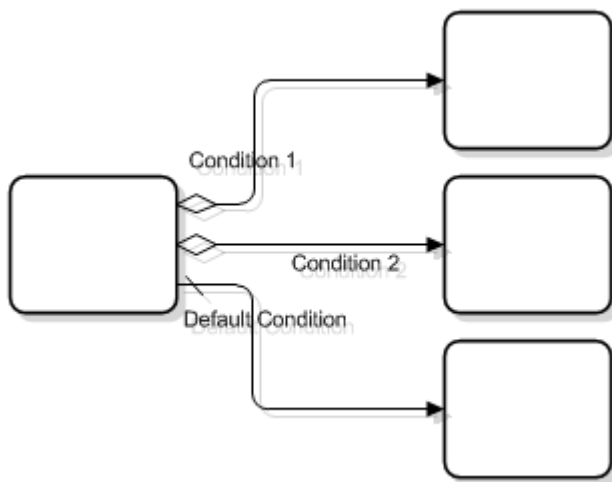


Figure 7.: An Inclusive Decision using Conditional Sequence Flow

There are some restrictions in using the conditional Sequence Flow (with mini-diamonds):

- The source object **MUST NOT** be an Event. The source object **MAY** a Gateway, but the mini-diamond **MUST NOT** be displayed in this case. The source object **MAY** be an activity (Task or Sub-Process) and the mini-diamond **SHALL** be displayed in this case.
  - A source Gateway **MUST NOT** be of type Parallel.
- If a conditional Sequence Flow is used from a source activity, then there **MUST** be at least one other outgoing Sequence Flow from that activity
  - The additional Sequence Flow(s) **MAY** also be conditional, but it is not required that they are conditional.

The second method for modeling Inclusive Decision situations uses an Inclusive Gateway, sometimes in combination with other Gateways. A marker will be placed in the center of the Gateway to indicate that the behavior of the Gateway is inclusive.

- The Inclusive Gateway **MUST** use a marker that is in the shape of a circle or an “O” and is placed within the Gateway diamond to distinguish it from other Gateways.

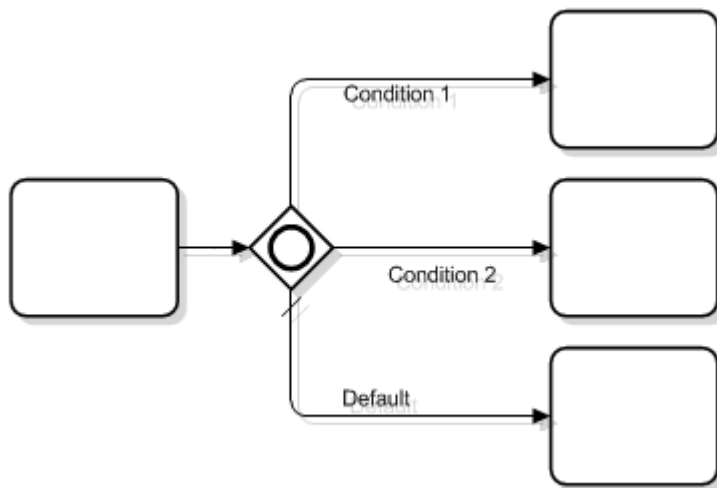


Figure 7.: An Inclusive Decision using an Inclusive Gateway

The behaviours of the two models depicted above are equivalent. Again, it is up to the modeler to insure that at least one of the conditions will be TRUE when the Process is performed.

When the Inclusive Gateway is used as a Merge, it will synchronize all Tokens that have been produced upstream, but at most one for each incoming Sequence Flow.

Note: Tokens with a loop are upstream of every node in the loop. It requires that Tokens for all Sequence Flow that were actually produced by an upstream (by an Inclusive situation, for example) be synchronized. If an upstream Inclusive produces two out of a possible three Tokens, then a downstream Inclusive will synchronize those two Tokens and not wait for another Token, even though there are three incoming Sequence Flow (see Figure below).

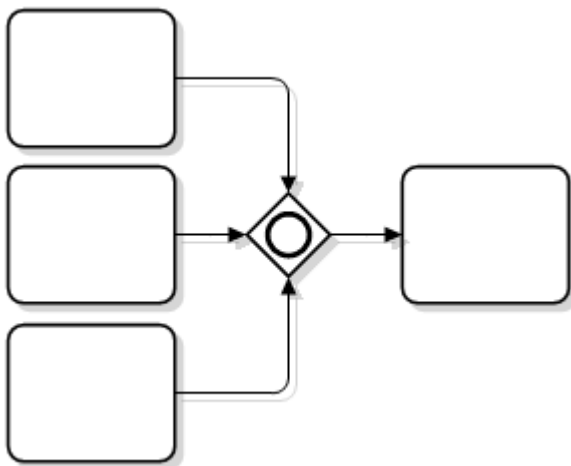


Figure 7.: An Inclusive Gateway Merging Sequence Flow

### SequenceFlow Connections for Inclusive Gateway

To define the inclusive nature of this Gateway's behavior for converging Sequence Flows:

- If there are multiple incoming Sequence Flows, one or more of them will be used to continue the flow of the Process. That is,
  - Process flow **SHALL** continue when the signals (Tokens) arrive from all of the incoming Sequence Flows that are expecting a signal based on the upstream structure of the Process (e.g., an upstream Inclusive Decision).
    - Some of the incoming Sequence Flow will not have signals and the pattern of which Sequence Flow will have signals may change for different instantiations of the Process.

**Note** – Incoming Sequence Flows that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flows from those Sequence Flows that have a source that is an upstream activity.

To define the inclusive nature of this Gateway's behavior for diverging Sequence Flows

- One or more Gates SHALL be selected during performance of the Process.
  - The Gates SHALL be chosen based on the Condition expression that is defined for the Sequence Flow associated with the Gates.
    - The Condition associated with all Gates SHALL be evaluated.
    - If a Condition is evaluated as “TRUE,” then that Gate SHALL be chosen, independent of what other Gates have or have not been chosen
    - If none of the ConditionExpressions for the Gates are evaluated as “TRUE,” then the DefaultGate shall be chosen.

#### 7.6.2.3.5. Complex Gateways

BPMN includes a Complex Gateway to handle situations that are not easily handled through the other types of Gateways. Complex Gateways can also be used to combine a set of linked simple Gateways into a single, more compact situation. Modelers can provide complex expressions that determine the merging and/or splitting behavior of the Gateway.

- The Complex Gateway MUST use a marker that is in the shape of an asterisk and is placed within the Gateway diamond (see Figure below) to distinguish it from other Gateways.

When the Gateway is used as a Decision (see Figure below), then an expression determines which of the outgoing Sequence Flow will be chosen for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flow. For example, an expression may evaluate Process data and then select different sets of outgoing Sequence Flow, based on the results of the evaluation. However, the expression should be designed so that at least one of the outgoing Sequence Flow will be chosen.

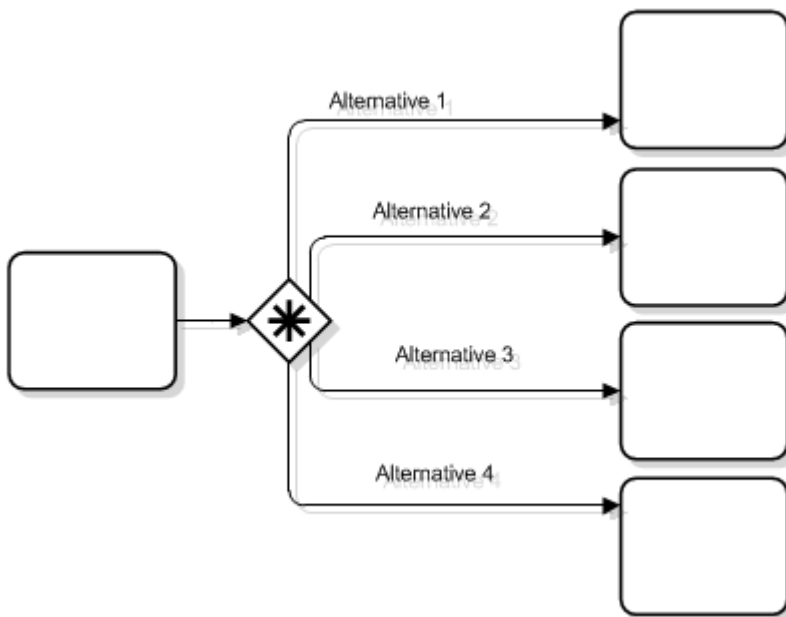


Figure 7.: A Complex Decision (Gateway)

When the Gateway is used as a Merge (see Figure below), then there will be an expression that will determine which of the incoming Sequence Flows will be required for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flows. For example, an expression may specify that any 3 out of 5 incoming Tokens will continue the Process. Another example would be an expression that specifies that a Token is required from Sequence Flow “a” and that a Token from either Sequence Flow “b” or “c” is acceptable. However, the expression should be designed so that the Process is not stalled at that location.

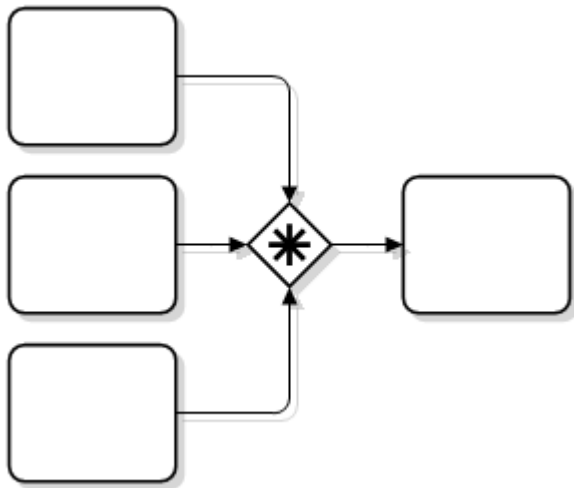


Figure 7.: A Complex Merge (Gateway)

#### Sequence Flow Connection Rules for a Complex Gateway

To define the complex nature of this Gateway's behavior for converging Sequence Flows:

- If there are multiple incoming Sequence Flows, one or more of them will be used to continue the flow of the Process. The exact combination of incoming Sequence Flows will be determined by the Gateway's IncomingCondition expression.
  - Process flow **SHALL** continue when the appropriate number of signals (Tokens) arrives from appropriate incoming Sequence Flows.
  - Signals from other Sequence Flows within that set **MAY** arrive, but they **MUST NOT** be used to continue the flow of the Process.

**Note** – Incoming Sequence Flows that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flows from those Sequence Flows that have a source that is an upstream activity.

To define the inclusive nature of this Gateway's behavior for diverging Sequence Flows

- One or more Gates **SHALL** be selected during performance of the Process.
  - The Gates **SHALL** be chosen based on the Gateway's OutgoingCondition expression.

#### 7.6.2.3.6. Parallel Gateways

Parallel Gateways provide a mechanism to synchronize parallel flows and to create parallel flows. These Gateways are not required to create parallel flows, but they can be used to clarify the behavior of complex situations where a string of Gateways are used and parallel flow is required. In addition, some modelers may wish to create a “best practice” where Parallel Gateways are always used for creating parallel paths. This practice will create an extra modeling element where one is not required, but will provide a balanced approach where forking and joining elements can be paired up.

- The Parallel Gateway **MUST** use a marker that is in the shape of a plus sign and is placed within the Gateway diamond to distinguish it from other Gateways.

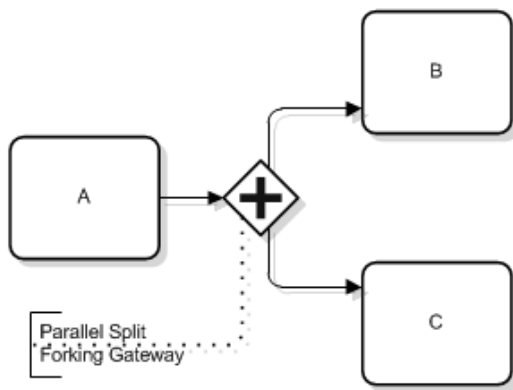


Figure 7.: A Parallel Gateway

Parallel Gateways are used for synchronizing parallel flow.

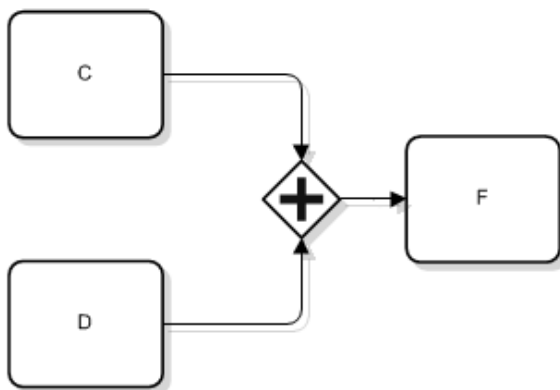


Figure 7.: Joining – the joining of parallel paths

### Sequence Flow Connections for Parallel Gateways

To define the parallel nature of this Gateway's behavior for converging Sequence Flows:

- If there are multiple incoming Sequence Flows, all of them will be used to continue the flow of the Process-- the flow will be synchronized. That is
  - Process flow **SHALL** continue when a signal (a Token) has arrived from all of a set of Sequence Flows (i.e., the process will wait for all signals to arrive before it can continue).

**Note** – Incoming Sequence Flows that have a source that is a downstream activity (that is, is part of a loop) will be treated differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flows from those Sequence Flows that have a source that is an upstream activity.

To define the parallel nature of this Gateway's behavior for diverging Sequence Flows:

- All Gates **SHALL** be selected during performance of the Process.

### 7.6.3. Block Activity/Embedded Sub-Process

A block activity executes an ActivitySet or self-contained activities/transitions map. From the Block Activity execution proceeds to the first activity in the set (unless otherwise specified by optional properties) and continues within the set until it reaches an exit activity (an activity with no output transitions). Execution then returns to follow the output transitions of the block activity.

A block activity/embedded subprocess call is transactional if the parent Activity element has been specified as transactional.

### 7.6.3.1. Schema for BlockActivity

	Description
ActivitySetId	The ActivitySet to be executed.
StartActivityId	If present, must be the id of a start activity in the ActivitySet referenced by the ActivitySetId attribute of BlockActivity. If not present then the start activity is deducible by some other means. See ActivitySet section 7.5.4.
View	Indicates whether the activity is COLLAPSED or EXPANDED.

Table BlockActivity

## 7.6.4. Event Activity

An Event is something that “happens” during the course of a business process. These Events affect the flow of the Process and usually have a cause (trigger or CATCH) or an impact (result or THROW). The term “event” is general enough to cover many things in a business process. The start of an activity, the end of an activity, the change of state of a document, a message that arrives, etc., all could be considered events. However, BPMN has restricted the use of events to include only those types of events that will affect the sequence or timing of activities of a process. BPMN further categorizes Events into three main types: Start, Intermediate, and End. Start and most Intermediate Events have “Triggers” that define the cause for the event. There are multiple ways that these events can be triggered. End Events may define a “Result” that is a consequence of a Sequence Flow ending. There are multiple types of Results that can be defined.

All Events share the same shape footprint, a small circle. Different line styles distinguish the three types of flow Events. All Events also have an open center so that BPMN-defined and modeler-defined icons can be included within the shape to help identify the Trigger or Result of the Event.



Figure 7.: Three Types of Events

### 7.6.4.1. Schema for Event

Schema contained in separate XSD.

Entity Types and Attributes(usage within EventType)	StartEvent	Intermediate Event	End Event	Reference
Activity				Used for Compensation. See section 7.6.4.5.1.
ErrorCode				See section 7.6.4.5.2.
Implementation	x	X		WebService   Other   Unspecified
Interrupting	x	x	x	Indicates whether event interrupts process
Name				See section 7.6.4.5.4.
Message	x	X	x	See section 7.9.4.

SubFlow/ProcessRef				See section 7.6.4.5.4.
Result			<b>x</b>	See section 7.6.4.4.5.
TriggerResultCompensation	<b>x</b>	<b>X</b>	<b>x</b>	See section 7.6.4.5.1.
ResultError	<b>x</b>	<b>X</b>	<b>x</b>	See section 7.6.4.5.2.
ResultMultiple			<b>x</b>	See section 7.6.4.5.3.
ConditionName	<b>x</b>	<b>x</b>		See section 7.6.4.5.10.
Target		<b>X</b>		A Target MAY be included for the Intermediate Event. The Target MUST be an activity. This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity. See section 7.6.4.3.
TimeCycle	<b>x</b>	<b>x</b>		See section 7.6.4.5.11.
TimeDate	<b>x</b>	<b>x</b>		See section 7.6.4.5.11.
Trigger	<b>x</b>	<b>X</b>		
TriggerIntermediateMultiple		<b>X</b>		See section 7.6.4.5.8.
TriggerMultiple	<b>x</b>			See section 7.6.4.5.9.
TriggerResultLink		<b>X</b>		See section 7.6.4.5.4.
TriggerResultMessage	<b>x</b>	<b>X</b>	<b>x</b>	See section 7.6.4.5.5.
TriggerResultCancel		<b>X</b>	<b>x</b>	
TriggerResultSignal	<b>x</b>	<b>X</b>	<b>x</b>	See section 7.6.4.5.7.
TriggerConditional	<b>x</b>	<b>X</b>		See section 7.6.4.5.10.
TriggerTimer	<b>x</b>	<b>X</b>		See section 7.6.4.5.11.
WebService	<b>x</b>	<b>X</b>	<b>x</b>	See section 7.9.6.

Table : Event

#### 7.6.4.2. Start Event

As the name implies, the Start Event indicates where a particular Process will start. In terms of Sequence Flow, the Start Event starts the flow of the Process, and thus, will not have any incoming Sequence Flow—no Sequence Flow can connect to a Start Event.

##### 7.6.4.2.1. Token Flow

To discuss Sequence Flow it is useful to refer to Petri Nets in general and in particular Place-Transition Nets and high-level nets such as Colored Petri Nets [High-Level Petri Nets, Springer-Verlag, 1991]. ‘Tokens’ traveling through such a structure characterize the ‘State’ of the system.

In our case a Token traverses the Sequence Flow and passes through the Flow Objects in the Process. Process behavior can then be described by the paths of the Token. To support parallel activity (caused by and-split (Fork) a token has a unique identity and ‘child-tokens’ with appropriate IDs are generated by the Fork.

A Start Event generates an instance of the Process. Thus a token is created at the Start Event node. This token travels through the network of Sequence Flow Activities and Gateways and is eventually consumed by an End Event. Arrival at

an And-Join Gateway requires multiple child tokens to be recombined. Arrival at an Intermediate Event requires the occurrence of the event (see discussion of Intermediate Event) in order for the token to exit that event. There should always be either Sequence Flow or a graphical indicator, such as an Intermediate Event to show all the potential paths of Tokens in Normal sequence Flow. An example of implicit flow is when a Token arrives at a Gateway, but none of the Gates are valid, the Token would then (implicitly) pass to the end of the Process, which occurs with some modeling notations. Tokens can also be directed through exception handling Intermediate Events, which act like a forced end to an activity.

Message flow can be modeled by a different class of tokens. We do not discuss this further in this specification.

#### 7.6.4.2.2. *Semantics of the Start Event*

Semantics of the Start Event include:

- A Start Event is OPTIONAL
  - If a Process is complex and/or the starting conditions are not obvious, then it is RECOMMENDED that a Start Event be used
  - If a BPMN process or subprocess does not have an explicit (drawn) Start Event, any BPMN activity or gateway with no incoming Sequence Flow (except for a Compensating Activity) is enabled when the process or subprocess is instantiated, i.e. is considered to include an implicit Start Event. Such implicit Start Events are not allowed if any explicit Start or End events are used in the same process or subprocess. A flow object with implicit Start Events shall have an implied trigger type of None, except for a Task with task type Receive or an Event Gateway.
  - If there is an End Event, then there MUST be at least one Start Event.
  - If the Start Event is used, then there MUST NOT be other flow elements that do not have incoming Sequence Flow—all other Flow Objects MUST be a target of at least one Sequence Flow.
    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation activities MUST NOT have any incoming Sequence Flow, even if there is a Start Event in the Process level.
    - An exception to this is the Intermediate Event, which MAY be without an incoming Sequence Flow (when attached to an activity boundary).
  - If the Start Event is not used, then all Flow Objects that do not have an incoming Sequence Flow (i.e., are not a target of a Sequence Flow) SHALL be instantiated when the Process is instantiated. There is an assumption that there is only one implicit Start Event, meaning that all the starting Flow Objects will start at the same time
    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities are not considered a part of the Normal Flow and MUST NOT be instantiated when the Process is instantiated.
- There MAY be multiple Start Events for a given Process level.
  - Each Start Event is an independent event. That is, a Process Instance SHALL be generated when the Start Event is triggered.
    - If the Process is used as a Sub-Process and there are multiple 'None' Start Events, then when flow is transferred from the parent Process to the Sub-Process, only one of the Sub-Process's Start Events will be Triggered. The TargetRef attribute of the Sequence Flow incoming to the Sub-Process object can be extended to identify the appropriate Start Event (as defined in the Sub-Process's "Sequence Flow Connections" on page 69). **[Note: XPD L provides a specific mechanism for choosing among multiple Start Activities. See StartActivityId in Process Definition 6.4.2]**

**Note** – The behavior of Process may be harder to understand if there are multiple Start Events. It is RECOMMENDED that this feature be used sparingly and that the modeler be aware that other readers of the Diagram may have difficulty understanding the intent of the Diagram.

When the trigger for a Start Event occurs, a new Process will be instantiated and a Token will be generated for each outgoing Sequence Flow from that event. The TokenId set for each of the Tokens will contain information such that the Tokens from the same parallel Fork **[AND-Split]** can be identified and the number of Tokens in the fork determined. These Tokens will begin their flow and not wait for any other Start Event to be triggered.

If there is a dependency for more than one Event to happen before a Process can start (e.g. two messages are required to start), then the Start Events must flow to the same activity within that Process. The attributes of the activity would specify when the activity could begin. If the attributes specify that the activity must wait for all inputs, then all Start



Events will have to be triggered before the Process begins. In addition, a correlation mechanism will be required so that different triggered Start Events will apply to the same process instance.








#### 7.6.4.2.3. Types of Start Events





There are many ways that a business process can be started (instantiated). The Trigger for a Start Event is designed to show the general mechanism that will instantiate that particular Process. In BPMN2.0 the introduction of event subprocesses further refines this mechanism and adds the notion of interrupting and non-interrupting events.

We subdivide our discussion into two sections: Top-level processes and Event sub-processes.

There are seven (7) types of Start Events for top-level Processes in BPMN (see Table 40): None, Message, Timer, Conditional, Signal, Multiple, and Parallel.

There are ten (10) types of StartEvents for Event Sub-Processes in BPMN (see Table 40): Message, Timer, Escalation, Error, Cancel, Compensation, Conditional, Signal, Multiple, and Parallel.

Trigger	Description (for Top Level Processes)	Marker
None	The modeler does not display the type of Event. It is also used for a Sub-Process that starts when the flow is triggered by sequenceflow from its Parent Process.	
Message	A Message arrives from a participant and triggers the start of the Process. The actual Participant from which the Message is received can be identified by connecting the Event to a Participant using a Message Flow.	
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process.	
Conditional	This type of event is triggered when a Condition such as “S&P 500 changes by more than 10% since opening,” or “Temperature above 300C” become true. The ConditionExpression for the Event must become false and then true before the Event can be triggered again.	
Signal	A signal arrives that has been broadcast from another Process and triggers the start of the Process. Note that the Signal is not a Message, which has a specific target for the Message. Multiple Processes can have Start Events that are triggered from the same broadcasted Signal.	
Multiple	This means that there are multiple ways of triggering the Process. Only one of them will be required to start the Process. The attributes of the Start Event will define which of the other types of Triggers apply.	
Parallel Multiple	This means that there are multiple triggers required before the Process can be instantiated. All of the types of triggers that are listed in the Start Event MUST be triggered before the Process is instantiated.	

Trigger	Description (for Event Sub- Processes)	Marker
Message	A Message arrives from a participant and triggers the start of the Process. The actual Participant from which the Message is received can be identified by connecting the Event to a Participant using a Message Flow. For a Message Event Sub-Process that interrupts its containing Process, the boundary of the Event is solid (see the upper figure to the right). For a Message Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process. For a <b>Timer Event Sub-Process</b> that interrupts its containing <b>Process</b> , the boundary of the <b>Event</b> is solid (see the upper figure to the right). For a <b>Timer Event Sub-Process</b> that does not interrupt its containing <b>Process</b> , the boundary of the <b>Event</b> is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting













Trigger	Description (for Event Sub- Processes)	Marker
Escalation	Escalation Event Sub-Processes implement measures to expedite the completion of a business activity, should it not satisfy a constraint specified on its execution (such as a time-based deadline). The Escalation Start Event is only allowed for triggering an in-line Event Sub-Process. For an Escalation Event Sub-Process that interrupts its containing Process, the boundary of the Event is solid (see the upper figure to the right). For an Escalation Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting
Error	The Error Start Event is only allowed for triggering an in-line Event Sub-Process. Given the nature of Errors, an Event Sub-Process with an Error trigger will always interrupt its containing Process.	 Interrupting
Compensation	The Compensation Start Event is only allowed for triggering an in-line Compensation Event Sub-Process. This type of Event is triggered when compensation occurs. Compensation Start Event uses a double triangle marker (see the figure to the right). This Event does not interrupt the Process since the Process has to be completed before this Event can be triggered.	 Non-Interrupting
Conditional	This type of event is triggered when a Condition such as “S&P 500 changes by more than 10% since opening,” or “Temperature above 300C” become true. The ConditionExpression for the Event must become false and then true before the Event can be triggered again. For a Conditional Event Sub-Process that interrupts its containing Process, then the boundary of the Event is solid (see the upper figure to the right). For a Conditional Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting
Signal	A signal arrives that has been broadcast from another Process and triggers the start of the Process. Note that the Signal is not a Message, which has a specific target for the Message. Multiple Processes can have Start Events that are triggered from the same broadcasted Signal. For a Signal Event Sub-Process that interrupts its containing Process, then the boundary of the Event is solid (see the upper figure to the right). For a Signal Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting
Multiple	A Multiple Event indicates that there are multiple ways of triggering the Event Sub-Process. Only one of them is required to actually start the Event Sub-Process. If the Start Event has more than one associated Event-Definiton, then the Event MUST be displayed with the Multiple Event marker (a pentagon—see the figures on the right). For a Multiple Event Sub-Process that interrupts its containing Process, the boundary of the Event is solid (see the upper figure to the right). For a Multiple Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting
Parallel Multiple	This means that there are multiple triggers required before the Process can be instantiated. All of the types of triggers that are listed in the Start Event MUST be triggered before the Process is instantiated. If the Start Event has more than one associated EventDefiniton and the parallelMultiple attribute of the Start Event is true, then the Event MUST be displayed with the Parallel Multiple Event marker (an open plus sign—see the figures to the right). For a Parallel Multiple Event Sub-Process that interrupts its containing Process, the boundary of the Event is solid (see the upper figure to the right). For a Parallel Multiple Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).	 Interrupting  Non-Interrupting

Table : Start Event subtypes

## 7.6.4.2.4. Start Event Sequence Flow Connections

- A Start Event MUST NOT be a target for Sequence Flow; it MUST NOT have incoming Sequence Flow.
  - An exception to this is when a Start Event is used in an Expanded Sub-Process and is attached to the boundary of that Sub-Process. In this case, a Sequence Flow from the higher-level Process MAY connect to that Start Event in lieu of connecting to the actual boundary of the Sub-Process. Note: this

is not necessary and inconsistent with the rest of the notation. It is not supported in XPDL.

- A Start Event **MUST** be a source for Sequence Flow.
- Multiple Sequence Flow **MAY** originate from a Start Event. For each Sequence Flow that has the Start Event as a source, a new parallel path **SHALL** be generated.
  - The Condition attribute for all outgoing Sequence Flows **MUST** be set to None.
  - When a Start Event is not used, then all Flow Objects that do not have an incoming Sequence Flow **SHALL** be the start of a separate parallel path.

Each path will have a separate unique Token that will traverse the Sequence Flow.

#### 7.6.4.2.5. Start Event Message Flow Connections

**Note** – All Message Flows must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- A Start Event **MAY** be the target for Message Flows; it can have 0 (zero) or more incoming Message Flows. Each Message Flow arriving at a Start Event represents an instantiation mechanism (a Trigger) for the process. Only one of the Triggers is required to start a new Process.
  - The Trigger attribute of the Start Event **MUST** be set to “Message” or “Multiple” if there are any incoming Message Flows.
    - The Trigger attribute of the Start Event **MUST** be set to “Multiple” if there are more than one incoming Message Flow.
- A Start Event **MUST NOT** be a source for Message Flow; it **MUST NOT** have outgoing Message Flows.

#### 7.6.4.2.6. Schema for StartEvent

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified See section 7.9.6.
Interrupting	Indicates whether the Event is interrupting
ResultError	The Error Start Event is only allowed for triggering an in-line Event Sub-Process.  Given the nature of Errors, an Event Sub-Process with an Error trigger will always interrupt its containing Process.
Trigger	None, Message, Timer.Error, Escallation, Compensation, Conditional, Signal, Multiple, ParallelMultiple. “None” is used typically for subflow invocations, including embedded subflows.
TriggerEscallation	Escalation Event Sub-Processes implement measures to expedite the completion of a business activity, should it not satisfy a constraint specified on its execution (such as a time-based deadline). The Escalation Start Event is only allowed for triggering an in-line Event Sub-ProcessFor an Escalation Event Sub-Process that interrupts its containing Process, the boundary of the Event is solid (see the upper figure to the right). For an Escalation Event Sub-Process that does not interrupt its containing Process, the boundary of the Event is dashed (see the lower figure on the right).
TriggerMultiple	This means that there are multiple ways of triggering the Process. Only one of them will be required to start the Process. The attributes of the Start Event will define which of the other types of Triggers apply. See section 7.6.4.5.9.
TriggerResultMessage	A message arrives from a participant and triggers the start of the Process. See section 7.6.4.5.5.
TriggerResultCompensat ion	The Compensation Start Event is only allowed for triggering an in-line Compensation Event Sub-Process. This type of Event is triggered when compensation occurs. Compensation Start Event uses a double triangle marker (see the figure to the right). This Event does not interrupt the Process since the Process has to be completed before this Event can be triggered.
TriggerConditional	This type of event is triggered when the conditions for a rule such as “S&P 500 changes by more than 10% since opening,” or “Temperature above 300C” become true. See section 7.6.4.5.10.

	Description
TriggerResultSignal	A signal arrives that has been broadcast from another Process and triggers the start of the Process. Note that the Signal is not a Message, which has a specific target for the Message. Multiple Processes can have Start Events that are triggered from the same broadcasted Signal.
TriggerTimer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process. See section 7.6.4.5.11.

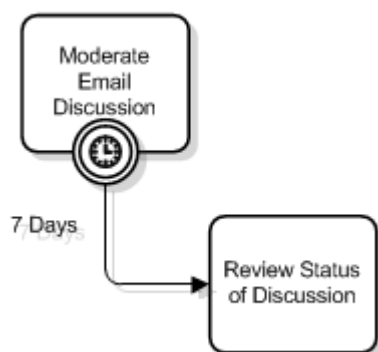
*Table : Start Event Activity*

#### 7.6.4.3. Intermediate Event

Intermediate Events occur between a Start Event and an End Event. This is an event that occurs after a Process has been started. It will affect the flow of the process, but will not start or (directly) terminate the process. Intermediate Events can be used to:

- Show where messages are expected or sent within the Process,
- Show where delays are expected within the Process,
- Disrupt the Normal Flow through exception handling, or
- Show the extra work required for compensation.

One use of Intermediate Events is to represent exception or compensation handling. This will be shown by placing the Intermediate Event on the boundary of a Task or Sub-Process (either collapsed or expanded). The figure below displays an example of an Intermediate Event attached to a Task. The Intermediate Event can be attached to any location of the activity boundary and the outgoing Sequence Flow can flow in any direction. However, in the interest of clarity of the Diagram, we recommend that the modeler choose a consistent location on the boundary. For example, if the Diagram orientation is horizontal, then the Intermediate Events can be attached to the bottom of the activity and the Sequence Flow directed down, then to the right. If the Diagram orientation is vertical, then the Intermediate Events can be attached to the left or right side of the activity and the Sequence Flow directed to the left or right, then down.



*Figure 7.: Intermediate Event on an Activity Boundary*

##### 7.6.4.3.1. Types of Intermediate Events

BPMN2.0 introduces the concept of interrupting and non-interrupting events as well as an additional event type.








There are twelve (12) types of Intermediate Events in BPMN: None, Message, Timer, Escalation, Error, Cancel, Compensation, Conditional, Link, Signal, **Multiple** and Parallel Multiple. Each type of Intermediate Event will have a different icon placed in the center of the Intermediate Event shape to distinguish one from another. An Intermediate Event that is placed within the normal flow of a Process can be used for one of two purposes. The Event can respond to (“catch”) the Event Trigger or the Event can be used to set off (“throw”) the Event Trigger. An Intermediate Event that is attached to the boundary of an Activity can only be used to “catch” the Event Trigger.








When a Token arrives at an Intermediate Event that is placed within the normal flow of a Process, one of two things will happen. If the Event is used to “throw” the Event Trigger, then Trigger of the Event will immediately occur (e.g., the Message will be sent) and the Token will move down the outgoing Sequence Flow. If the Event is used to “catch” the Event Trigger, then the Token will remain at the Event until the Trigger occurs (e.g., the Message is received). Then the Token will move down the outgoing Sequence Flow.





We subdivide our discussion into two sections: Intermediate Events within the normal flow and Boundary Events.










There are ten (10) types of Intermediate Events within the normal flow (see Table 42): None, Message, Timer, Escalation, Compensation, Conditional, Link, Signal, Multiple, and Parallel Multiple.





There are ten (10) types of Boundary Events (see Table 42): Message, Timer, Escalation, Error, Cancel, Compensation, Conditional, Signal, Multiple, and Parallel Multiple.

Trigger	Description for Intermediate Events in Normal Flow	Marker
None	The None Intermediate Event is only valid in Normal Flow, i.e., it may not be used on the boundary of an Activity. Although there is no specific trigger for this Event, it is defined as throw Event. It is used for modeling methodologies that use Events to indicate some change of state in the Process.	 Throw
Message	A Message Intermediate Event can be used to either send a Message or receive a Message. When used to “throw” the message, the Event marker MUST be filled (see the upper figure on the right). When used to “catch” the message, then the Event marker MUST be unfilled (see the lower figure on the right). This causes the Process to continue if it was waiting for the message, or changes the flow for exception handling. The actual Participant from which the Message is received can be identified by connecting the Event to a Participant using a Message Flow.	 Throw  Catch
Timer	In Normal Flow the Timer Intermediate Event acts as a delay mechanism based on a specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the Event. This Event MUST be displayed with a clock marker (see the figure on the right).	 Catch
Escalation	In Normal Flow, the Escalation Intermediate Event raises an Escalation. Since this is a Throw Event, the arrowhead marker will be filled (see the figure to the right).	 Throw
Compensation	In normal flow, this Intermediate Event indicates that compensation is necessary. Thus, it is used to “throw” the Compensation Event, and the Event marker MUST be filled (see figure on the right). If an Activity is identified, and it was successfully completed, then that Activity will be compensated. The activity must be visible from the Compensation Intermediate Event, i.e., one of the following must be true: <ul style="list-style-type: none"> <li>The Compensation Intermediate Event is contained in normal flow at the same level of Sub-Process.</li> <li>The Compensation Intermediate Event is contained in a Compensation Event Sub-Process which is contained in the Sub-Process containing the Activity.</li> </ul> If no Activity is identified, all successfully completed Activities visible from the Compensation Intermediate Event are compensated, in reverse order of their Sequence Flow. Visible means one of the following: <ul style="list-style-type: none"> <li>The Compensation Intermediate Event is contained in normal flow and at the same level of Sub-Process as the Activities.</li> <li>The Compensation Intermediate Event is contained in a Compensation Event Sub-Process which is contained in the Sub- Process containing the Activities.</li> </ul> To be compensated, an Activity MUST have a boundary Compensation Event, or contain a Compensation Event Sub-Process.	 Throw
Conditional	This type of event is triggered when a Condition becomes true.	 Catch

Trigger	Description for Intermediate Events in Normal Flow	Marker
Link	The Link Intermediate Events are only valid in Normal Flow, i.e. they may not be used on the boundary of an Activity. A Link is a mechanism for connecting two sections of a Process. Link Events can be used to create looping situations or to avoid long Sequence Flow lines. Link Event uses are limited to a single Process level (i.e., they cannot link a parent Process with a Sub-Process). Paired Intermediate Events can also be used as “Off-Page Connectors” for printing a Process across multiple pages. They can also be used as generic “Go To” objects within the Process level. There can be multiple Source Link Events, but there can only be one Target Link Event. When used to “throw” to the Target Link, the Event marker will be filled (see the top figure on the right). When used to “catch” from the Source Link, the Event marker will be unfilled (see the bottom figure on the right).	 Throw  Catch
Signal	This type of event is used for sending or receiving Signals. A Signal is for general communication within and across Process Levels, across Pools, and between Business Process Diagrams. A BPMN Signal is similar to a signal flare that shot into the sky for anyone who might be interested to notice and then react. Thus, there is a source of the Signal, but no specific intended target. This type of Intermediate Event can send or receive a Signal if the Event is part of a Normal Flow. The Signal Event differs from an Error Event in that the Signal defines a more general, nonerror condition for interrupting activities (such as the successful completion of another activity) as well as having a larger scope than Error Events. When used to “catch” the signal, the Event marker will be unfilled. When used to “throw” the signal, the Event marker will be filled.	 Throw  Catch
Multiple	This means that there are multiple Triggers assigned to the Event. If used within normal flow, the Event can “catch” the Trigger or “throw” the Triggers. When used to “catch” the Trigger, only one of the assigned Triggers is required and the Event marker will be unfilled. When used to “throw” the Trigger (the same as a Multiple End Event), all the assigned Triggers will be thrown.	 Throw  Catch
Parallel Multiple	This means that there are multiple triggers assigned to the Event. If used within normal flow, the Event can only “catch” the trigger. When attached to the boundary of an activity, the Event can only “catch” the trigger. Unlike the normal Multiple Intermediate Event, all of the assigned triggers are required for the Event to be triggered.	 Catch

Trigger	Description for Intermediate Boundary Events	Marker
Message	A Message arrives from a participant and triggers the Event. If a Message Event is attached to the boundary of an Activity, it will change the Normal Flow into an Exception Flow upon being triggered. For a Message Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true. For a Message Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false. The actual Participant from which the Message is received can be identified by connecting the Event to a Participant using a Message Flow	 Interrupting  Non-Interrupting
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the Event. If a Timer Event is attached to the boundary of an Activity, it will change the Normal Flow into an Exception Flow upon being triggered. For a Timer Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true. For a Timer Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.	 Interrupting  Non-Interrupting

Trigger	Description for Intermediate Boundary Events	Marker
Escalation	<p>This type of Event is used for handling a named Escalation. If attached to the boundary of an activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the activity to which the boundary event is attached. However, a modeler may decide to override this setting by using the notation described in the following.</p> <p>For an Escalation Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true.</p> <p>For an Escalation Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.</p>	 Interrupting  Non-Interrupting
Error	<p>An Intermediate Error Catch Event can only be attached to the boundary of an activity, i.e., it may not be used in Normal Flow. If used in this context, it reacts to (catches) a named error, or to any error if a name is not specified. Note that an Error Event always interrupts the Activity to which it is attached, i.e., there is not a non-interrupting version of this Event. The boundary of the Event thus always solid (see figure on the right).</p>	 Interrupting
Cancel	<p>This type of Intermediate Event is used within a Transaction Sub-Process. This type of Event MUST be attached to the boundary of a Sub-Process. It SHALL be triggered if a Cancel End Event is reached within the Transaction Sub-Process. It also SHALL be triggered if a Transaction Protocol "Cancel" message has been received while the Transaction is being performed. Note that a Cancel Event always interrupts the Activity to which it is attached, i.e., there is not a non-interrupting version of this Event. The boundary of the Event thus always solid (see figure on the right).</p>	 Interrupting
Compensation	<p>When attached to the boundary of an Activity, this Event is used to "catch" the Compensation Event, thus the Event marker MUST be unfilled (see figure on the right). The Event will be triggered by a thrown compensation targeting that Activity. When the Event is triggered, the Compensation Activity that is Associated to the Event will be performed.</p> <p>Note that the interrupting or non-interrupting aspect of other Events does not apply in the case of a Compensation Event. Compensations can only be triggered after completion of the activity to which they are attached. Thus they cannot interrupt the Activity. The boundary of the Event is always solid.</p>	 Interrupting
Conditional	<p>This type of event is triggered when a Condition becomes true. A Condition is a type of Expression. If a Conditional Event is attached to the boundary of an Activity, it will change the normal flow into an exception flow upon being triggered. For a Conditional Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true.</p> <p>For a Conditional Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.</p>	 Interrupting  Non-Interrupting
Signal	<p>The Signal Event can only receive a Signal when attached to the boundary of an activity. In this context, it will change the Normal Flow into an Exception Flow upon being triggered. The Signal Event differs from an Error Event in that the Signal defines a more general, non-error condition for interrupting activities (such as the successful completion of another activity) as well as having a larger scope than Error Events. When used to "catch" the signal, the Event marker will be unfilled.</p> <p>For a Signal Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true.</p> <p>For a Signal Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.</p>	 Interrupting  Non-Interrupting

Trigger	Description for Intermediate Boundary Events	Marker
Multiple	<p>A Multiple Event indicates that there are multiple Triggers assigned to the Event. When attached to the boundary of an activity, the Event can only “catch” the Trigger. In this case, only one of the assigned Triggers is required and the Event marker will be unfilled. Upon being triggered, the Event that occurred will change the Normal Flow into an Exception Flow.</p> <p>For a Multiple Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see upper figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true.</p> <p>For a Multiple Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.</p>	 Interrupting  Non-Interrupting
Parallel Multiple	<p>This means that there are multiple triggers assigned to the Event. When attached to the boundary of an activity, the Event can only “catch” the trigger. Unlike the normal Multiple Intermediate Event, all of the assigned triggers are required for the Event to be triggered. The Event marker will be an unfilled plus sign (see the figures on the right).</p> <p>For a Parallel Multiple Event that interrupts the Activity to which it is attached, the boundary of the Event is solid (see the upper figure to the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true.</p> <p>For a Parallel Multiple Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see the lower figure to the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.</p>	 Interrupting  Non-Interrupting

*Table : Intermediate Event sub types*

#### 7.6.4.3.2. Activity Boundary Connections

An Intermediate Event can be attached to the boundary of an activity under the following conditions:

- (One or more) Intermediate Events MAY be attached directly to the boundary of an Activity.
  - To be attached to the boundary of an Activity, an Intermediate Event MUST be one of the following Triggers: Message, Timer, Error, Escalation, Cancel, Compensation, Conditional, Signal, Multiple or Parallel Multiple.
    - An Intermediate Event with a Cancel Trigger MAY be attached to a Sub-Process boundary only if the IsATransaction attribute of the Sub-Process is set to TRUE.

#### 7.6.4.3.3. Intermediate Event Sequence Flow Connections

- If the Intermediate Event is attached to the boundary of an activity:
  - The Intermediate Event MUST NOT be a target for Sequence Flow; it cannot have an incoming Flow
  - The Intermediate Event MUST be a source for Sequence Flow; it can have one or more outgoing Sequence Flow. For each Sequence Flow that has the Intermediate Event as a source, a new parallel path SHALL be generated.
    - An exception to this: an Intermediate Event with a Compensation Trigger MUST NOT have an outgoing Sequence Flow (it MAY have an outgoing Association).
- If the Intermediate Event is used within Normal Flow:
  - Intermediate Events MUST be a target of a Sequence Flow; they MUST have at least one incoming Flow.

**Note** –If the Event has multiple incoming Sequence Flows, then this is considered uncontrolled flow. This means that when a token arrives from one of the Paths, the Event will be enabled (to catch or throw). It will not wait for the arrival of tokens from the other paths. If another token arrives from the same path or another path, then a separate instance of the Event will be created. If the flow needs to be controlled, then the flow should converge with a Gateway that precedes the Event.



- An Intermediate Event **MUST** be a source for Sequence Flow.
- Multiple Sequence Flow **MAY** originate from an Intermediate Event. For each Sequence Flow that has the Intermediate Event as a source, a new parallel path **SHALL** be generated.
  - An exception to this: a Source Link Intermediate Event (as defined below), is not required to have an outgoing Sequence Flow.
- An Intermediate Event with a Link Trigger **MUST NOT** be both a target and a source of a Sequence Flow.

To define the use of a Link Intermediate Event as an “Off-Page Connector” or a “Go To” object:

- A Link Intermediate Event **MAY** be the target (Target Link) or a source (Source Link) of a Sequence Flow, but **MUST NOT** be both a target and a source.
  - If there is a Source Link, there **MUST** be a matching Target Link (they have the same Name).
    - There **MAY** be multiple Source Links for a single Target Link.
    - There **MUST NOT** be multiple Target Links for a single Source Link.

#### 7.6.4.3.4. *Intermediate Event Message Flow Connections*

**Note** – All Message Flows must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- An Intermediate Event of type Message **MAY** be the target for Message Flow; it can have one incoming Message Flow.
- An Intermediate Event of type Message **MAY** be a source for Message Flow; it can have one outgoing Message Flow.
- An Intermediate Event of type Message **MAY** have an incoming Message Flow or an outgoing Message Flow, but not both.

#### 7.6.4.3.5. *Schema for Intermediate Event*

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified See 7.9.6.
Interrupting	Determines if Event is interrupting.
TriggerResultCompensation	This is used for compensation handling--both setting and performing compensation. It calls for compensation if the Event is part of a Normal Flow. It reacts to a named compensation call when attached to the boundary of an activity. See 7.6.4.5.1.
ResultError	This is used for error handling--both to set (throw) and to react to (catch) errors. It sets (throws) an error if the Event is part of a Normal Flow. It reacts to (catches) a named error, or to any error if a name is not specified, when attached to the boundary of an activity. See 7.6.4.5.2.
Target	A Target <b>MAY</b> be included for the Intermediate Event. The Target <b>MUST</b> be an activity (Sub-Process or Task). This means that the Intermediate Event is attached to the boundary of the activity and is used to signify an exception or compensation for that activity.
Trigger	None, Message, Timer, Error Escalation, Cancel, Conditional, Link, Signal, Compensation, Multiple, ParallelMultiple. The None and Link Trigger <b>MUST NOT</b> be used when the Event is attached to the boundary of an Activity. The Multiple, Rule, and Cancel Triggers <b>MUST NOT</b> be used when the Event is part of the Normal Flow of the Process. The Cancel Trigger <b>MUST NOT</b> be used when the Event is attached to the boundary of a Transaction or if the Event is not contained within a Process that is a Transaction. “None” is used typically for Intermediate Events that are in the main flow of the Process. It is used for modeling methodologies that use Events to indicate some change of state in the Process.

	Description
TriggerCancel	This type of Intermediate Event is used within a Transaction Sub-Process. This type of Event MUST be attached to the boundary of a Sub-Process. It SHALL be triggered if a Cancel End Event is reached within the Transaction Sub-Process. It also SHALL be triggered if a Transaction Protocol “Cancel” message has been received while the Transaction is being performed.
TriggerIntermediateMultiple	This means that there are multiple triggers listed. See 7.6.4.5.9
TriggerResultCancel	This type of Intermediate Event is used within a Transaction Sub-Process. This type of Event MUST be attached to the boundary of a Sub- Process. It SHALL be triggered if a Cancel End Event is reached within the Transaction Sub-Process. It also SHALL be triggered if a Transaction Protocol “Cancel” message has been received while the Transaction is being performed. Note that a Cancel Event always interrupts the Activity to which it is attached, i.e., there is not a non-interrupting version of this Event. The boundary of the Event thus always solid (see figure on the right).
TriggerResultLink	If the Trigger is a Link, then the Name MUST be supplied. See 7.6.4.5.4
TriggerResultMessage	A message arrives from a participant and triggers the Event. This causes the Process to continue if it was waiting for the message, or changes the flow for exception handling. In Normal Flow, Message Intermediate Events can be used for sending messages to a participant. If used for exception handling it will change the Normal Flow into an Exception Flow. See section 7.6.4.5.5.
TriggerConditional	This type of event is triggered when the conditions for a rule such as “S&P 500 changes by more than 10% since opening,” or “Temperature above 300C” become true. See section 7.6.4.5.10.
TriggerEscalation	This type of Event is used for handling a named Escalation. In Normal Flow, the Escalation Intermediate Event raises an Escalation. If attached to the boundary of an activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the activity to which the boundary event is attached. However, a modeler may decide to override this setting by using the notation described in the following. For an Escalation Event that interrupts the Activity to which it is attached, the boundary of the Event is solid. Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to true. For an Escalation Event that does not interrupt the Activity to which it is attached, the boundary of the Event is dashed (see lower figure on the right). Note that if using this notation, the attribute cancel Activity of the Activity to which the Event is attached is implicitly set to false.
TriggerResultSignal	This type of event is used for sending or receiving Signals. A Signal is for general communication within and across Process Levels, across Pools, and between Business Process Diagrams. A BPMN Signal is similar to a signal flare that shot into the sky for anyone who might be interested to notice and then react. Thus, there is a source of the Signal, but no specific intended target. This is different than a BPMN Message, which has a specific Source and a specific Target (which can be an Entity or an abstract Role). This type of Intermediate Event can send or receive a Signal if the Event is part of a Normal Flow. The Event can only receive a Signal when attached to the boundary of an activity. The Signal Event differs from an Error Event in that the Signal defines a more general, non-error condition for interrupting activities (such as the successful completion of another activity) as well as having a larger scope than Error Events. When used to “catch” the signal, the Event marker will be unfilled (see the top figure on the right). When used to “throw” the signal, the Event marker will be filled.
TriggerTimer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the Event. If used within the main flow it acts as a delay mechanism. If used for exception handling it will change the Normal Flow into an Exception Flow. See section 7.6.4.5.11.

Table : Intermediate Event Activity

**7.6.4.4. End Event**

As the name implies, the End Event indicates where a process will end. In terms of Sequence Flow, the End Event ends the flow of the Process, and thus, will not have any outgoing Sequence Flow—no Sequence Flow can connect from an End Event.

An End Event consumes a Token that had been generated from a Start Event within the same level of Process. If parallel Sequence Flows target the End Event, then the Tokens will be consumed as they arrive. All the Tokens that were generated within the Process must be consumed by an End Event before the Process has been completed. In other circumstances, if the Process is a Sub-Process, it can be stopped prior to normal completion through interrupting Intermediate Events. In this situation the Tokens will be consumed by an Intermediate Event attached to the boundary of the Sub-Process.





#### 7.6.4.4.1. *Semantics of the End Event*

- There MAY be multiple End Events within a single level of a process.
- An End Event is OPTIONAL: a given Process level—a top-level Process or an expanded Sub-Process—MAY (is not required to) have this shape:
  - If an End Event is not used, then the implicit End Event for the Process SHALL NOT have a Result.
  - If there is a Start Event, then there MUST be at least one End Event.
  - If an End Event is used, then there MUST NOT be other flow elements that do not have any outgoing Sequence Flow—all other Flow Objects MUST be a source of at least one Sequence Flow.
    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities MUST NOT have any outgoing Sequence Flow, even if there is an End Event in the Process level.
  - If the End Event is not used, then all Flow Objects that do not have any outgoing Sequence Flow (i.e., are not a source of a Sequence Flow) mark the end of a path in the Process. However, the process MUST NOT end until all parallel paths have completed.
    - Exceptions to this are activities that are defined as being Compensation activities (have the Compensation Marker). Compensation Activities are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

For Processes without an End Event, a Token entering a path-ending Flow Object will be consumed when the processing performed by the object is completed (i.e., when the path has completed), as if the Token had then gone on to reach an End Event. When all Tokens for a given instance of the Process are consumed, then the Process will reach a state of being completed.

#### 7.6.4.4.2. *Types of End Events*

There are eight (8) types of End Events in BPMN: None, Message, Escalation, Error, Cancel, Compensation, Signal, Terminate, and Multiple. These types define the consequence of reaching an End Event. This will be referred to as the End Event Result.

Result	Description	Marker
None	The modeler does not display the type of Event. It is also used to show the end of a Sub-Process that ends, which causes the flow to go back to its Parent Process.	
Message	This type of End indicates that a message is sent to a participant at the conclusion of the Process.	
Error	This type of End indicates that a named Error should be generated. All currently active threads in the particular Sub-Process are terminated as a result. The Error will be caught by a Catch Error intermediate event with the same ErrorCode or no ErrorCode which is on the boundary of the nearest enclosing parent activity (hierarchically). The behavior of the process is unspecified if no enclosing parent activity has such an Error intermediate event. The system executing the process may define additional Error handling in this case, a common one being termination of the process instance.	
Escalation	This type of End indicates that an Escalation should be triggered. Other active threads are not affected by this and continue to be executed. The Escalation will be caught by a Catch Escalation Intermediate Event with the same escalationCode or	






Result	Description	Marker
	no escalationCode which is on the boundary of the nearest enclosing parent activity (hierarchically). The behavior of the process is unspecified if no activity in the hierarchy has such an Escalation Intermediate Event.	
Cancel	This type of End is used within a Transaction Sub-Process. It will indicate that the Transaction should be cancelled and will trigger a Cancel Intermediate Event attached to the Sub-Process boundary. In addition, it will indicate that a Transaction Protocol Cancel message should be sent to any Entities involved in the Transaction.	
Compensation	This type of End indicates that compensation is necessary. If an Activity is identified, and it was successfully completed, then that Activity will be compensated. The Activity must be visible from the Compensation End Event, i.e., one of the following must be true: <ul style="list-style-type: none"> <li>• The Compensation End Event is contained in normal flow at the same level of Sub-Process.</li> <li>• The Compensation End Event is contained in a Compensation Event Sub-Process which is contained in the Sub-Process containing the Activity.</li> <li>• If no Activity is identified, all successfully completed Activities visible from the Compensation End Event are compensated, in reverse order of their Sequence Flow.</li> </ul> Visible means one of the following: <ul style="list-style-type: none"> <li>• The Compensation End Event is contained in normal flow and at the same level of Sub-Process as the Activities.</li> <li>• The Compensation End Event is contained in a Compensation Event Sub-Process which is contained in the Sub-Process containing the Activities.</li> </ul> To be compensated, an Activity MUST have a boundary Compensation Event or contain a Compensation Event Sub-Process	
Signal	This type of End indicates that a Signal will be broadcasted when the End has been reached. Note that the Signal, which is broadcast to any Process that can receive the Signal, can be sent across Process levels or Pools, but is not a Message (which has a specific Source and Target).	
Terminate	This type of End indicates that all activities in the Process should be immediately ended. This includes all instances of Multi-Instances. The Process is ended without compensation or event handling.	
Multiple	This means that there are multiple consequences of ending the Process. All of them will occur (e.g., there might be multiple messages sent). The attributes of the End Event will define which of the other types of Results apply.	

Table : End Event subtypes

#### 7.6.4.4.3. End Event Sequence Flow Connections

- An End Event MUST be a target for Sequence Flow.
- An End Event MAY have multiple incoming Sequence Flows.

The Flow MAY come from either alternative or parallel paths. For modeling convenience, each path MAY connect to a separate End Event object. The End Event is used as a Sink for all Tokens that arrive at the Event. All Tokens that are generated at the Start Event for that Process must eventually arrive at an End Event. The Process will be in a running state until all Tokens are consumed.

- An End Event MUST NOT be a source for Sequence Flow; that is, there MUST NOT be outgoing Sequence Flow.
  - An exception to this is when an End Event is used in an Expanded Sub-Process and is attached to the boundary of that Sub-Process. In this case, a Sequence Flow from the higher-level Process MAY connect from that End Event in lieu of connecting from the actual boundary of the Sub-Process. Note: XPDL does not support an End Event attached to the Sub-Process boundary.

#### 7.6.4.4.4. End Event Message Flow Connections

**Note** – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- An End Event MUST NOT be the target for Message Flow; it can have no incoming Message Flow. If the Intermediate Event has an incoming Message Flow, then it MUST NOT have an outgoing Message Flow.

- An Intermediate Event of type Message, if it is used within Normal Flow, MAY be the source for Message Flow; it can have one outgoing Message Flow. If the Intermediate Event has an outgoing Message Flow, then it MUST NOT have an incoming Message Flow.

#### 7.6.4.4.5. Schema for End Event

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified See 7.9.6.
TriggerEscalation	This type of End indicates that an Escalation should be triggered. Other active threads are not affected by this and continue to be executed. The Escalation will be caught by a Catch Escalation Intermediate Event with the same escalationCode or no escalationCode which is on the boundary of the nearest enclosing parent activity (hierarchically). The behavior of the process is unspecified if no activity in the hierarchy has such an Escalation Intermediate Event.
Result (fixed to be required)	None, Message, Error, Cancel, Compensation, Signal, Terminate, Multiple.  ‘None’ is used when the modeler does not display the type of Event. It is also used to show the end of a Sub-Process that ends, which causes the flow goes back to its Parent Process.  “Terminate” indicates that all activities in the Process should be immediately ended. This includes all instances of Multi-Instances. The Process is ended without compensation or event handling.
TriggerResultCancel	This type of End is used within a Transaction Sub-Process. It will indicate that the Transaction should be cancelled and will trigger a Cancel Intermediate Event attached to the Sub-Process boundary. In addition, it will indicate that a Transaction Protocol Cancel message should be sent to any Entities involved in the Transaction.
TriggerResultCompensation	This type of End will indicate that a Compensation is necessary. The Compensation identifier will trigger an Intermediate Event when the Process is rolling back. See section 7.6.4.5.1,
TriggerResultMessage	This type of End indicates that a message is sent to a participant at the conclusion of the Process.
TriggerResultSignal	This type of End indicates that a Signal will be broadcasted when the End has been reached. Note that the Signal, which is broadcast to any Process that can receive the Signal, can be sent across Process levels or Pools, but is not a Message (which has a specific Source and Target).
TriggerResultTerminate	This type of End indicates that all activities in the Process should be immediately ended. This includes all instances of Multi-Instances. The Process is ended without compensation or event handling.
ResultError	This type of End indicates that a named Error should be generated. This Error will be caught by an Intermediate Event within the Event Context. See section 7.6.4.5.2.
ResultMultiple	This means that there are multiple consequences of ending the Process. All of them will occur (e.g., there might be multiple messages sent). The attributes of the End Event will define which of the other types of Results apply. See section 7.6.4.5.3.

Table : End Event Activity

#### 7.6.4.5. Common Elements Used in Start, Intermediate and End Events

##### 7.6.4.5.1. TriggerResultCompensation

For an End Event: If the Result is a Compensation, then the Activity that needs to be compensated MAY be supplied. If an Activity is not supplied, then the Event is broadcast to all completed activities in the Process Instance.

For an Intermediate Event within Normal Flow: If the Trigger is a Compensation, then the Activity that needs to be compensated MAY be supplied. If an Activity is not supplied, then the Event is broadcast to all completed activities in the Process Instance. This “throws” the compensation.

For an Intermediate Event attached to the boundary of an Activity: This Event “catches” the compensation. No further information is required. The Id of the activity the Event is attached to will provide the Id necessary to match the

compensation event with the event that “threw” the compensation.

Compensation requires specific notation and is a special circumstance that occurs outside the Normal Flow of the Process. For this reason, the Compensation Intermediate Event does not have an outgoing Sequence Flow, but instead has an outgoing directed Association. The target of this Association is the activity that will compensate for the work done in the source activity, and will be referred to as the Compensation Activity. The Compensation Activity is special in that it does not follow the normal Sequence Flow rules--as mentioned; it is outside the Normal Flow of the Process. This activity cannot have any incoming or outgoing Sequence Flow. The Compensation marker (as is in the Compensation Intermediate Event) will be displayed in the bottom center of the Activity to show this status of the activity.

Schema in XSD file.

	Description
ActivityId	<p><b>For a Start Event:</b> This Event “catches” the compensation for an Event Sub-Process. No further information is required. The Event Sub-Process will provide the Id necessary to match the Compensation Event with the Event that threw the compensation, or the compensation will have been a broadcast.</p> <p><b>For an End Event:</b> The Activity to be compensated MAY be supplied. If an Activity is not supplied, then the compensation is broadcast to all completed Activities in the current Sub-Process (if present), or the entire Process Instance (if at the global level).</p> <p><b>For an Intermediate Event within Normal Flow:</b> The Activity to be compensated MAY be supplied. If an Activity is not supplied, then the compensation is broadcast to all completed Activities in the current Sub-Process (if present), or the entire Process Instance (if at the global level). This “throws” the compensation.</p> <p><b>For an Intermediate Event attached to the boundary of an Activity:</b> This Event “catches” the compensation. No further information is required. The Activity the Event is attached to will provide the Id necessary to match the Compensation Event with the Event that threw the compensation, or the compensation will have been a broadcast.</p>
WaitForCompletion	For a throw Compensation Event, this flag determines whether the throw Intermediate Event waits for the triggered compensation to complete (the default), or just triggers the compensation and immediately continues (the BPMN 1.2 behavior).

*Table : Event Trigger Result Compensation*

#### 7.6.4.5.2. ResultError

This is used for error handling--both to set (throw) and to react to (catch) errors. It sets (throws) an error if the Event is part of a Normal Flow (either Intermediate or End Event). It reacts to (catches) a named error, or to any error if a name is not specified, when attached to the boundary of an activity.

For an End Event: If the Result is an Error, then the ErrorCode MUST be supplied. This “throws” the error.

For an Intermediate Event within Normal Flow: If the Trigger is an Error, then the ErrorCode MUST be entered. This “throws” the error.

For an Intermediate Event attached to the boundary of an Activity: If the Trigger is an Error, then the ErrorCode MAY be entered. This Event “catches” the error. If there is no ErrorCode, then any error SHALL trigger the Event. If there is an ErrorCode, then only an error that matches the ErrorCode SHALL trigger the Event.

Schema in XSD file.

	Description
ErrorCode	String designates the error Code.

*Table : Event Result Error*

#### 7.6.4.5.3. ResultMultiple

For an End Event:

This means that there are multiple consequences of ending the Process. All of them will occur (e.g., there might be multiple messages sent). The attributes of the End Event will define which of the other types of Results apply.

Schema in XSD file.

	Description
	TriggerResultMessage, TriggerEscalation, TriggerResultCancel, TriggerResultCompensation, TriggerResultSignal, TriggerResultTerminate, ResultError.

*Table : Event Result Multiple*

#### 7.6.4.5.4. *TriggerResultLink*

For an Intermediate Event:

Paired Intermediate Events can be used as “Go To” objects within a Process.

For all Event Types:

The Name **MUST** be entered.

Schema is in XSD file.

	Description
Name	Unique name for this link.

*Table : Event Trigger Result Link*

#### 7.6.4.5.5. *TriggerResultMessage*

For Start Event:

A message arrives [CATCH] and starts the Process.

For End Event:

This type of End indicates that a message is sent [THROW] at the conclusion of the Process.

For Intermediate Event:

A message arrives from a participant and triggers the Event. This causes the Process to continue if it was waiting for the message, or changes the flow for exception handling. In Normal Flow, Message Intermediate Events can be used for sending messages to a participant. If used for exception handling it will change the Normal Flow into an Exception Flow.

Schema in XSD file.

	Description
Message	Describes the message (See section 7.9.4).
WebServiceOperation	Describes the web services operation (See section 7.9.6)

*Table : Event Trigger Result Message*

#### 7.6.4.5.6. *TriggerResultCancel*

A Cancel Intermediate Event is used within a Transaction Sub-Process. This type of Event **MUST** be attached to the boundary of a Sub-Process. It **SHALL** be triggered if a Cancel End Event is reached within the Transaction Sub-Process. It also **SHALL** be triggered if a Transaction Protocol “Cancel” message has been received while the Transaction is being performed.

A Cancel End Event is used within a Transaction Sub-Process. It will indicate that the Transaction should be cancelled and will trigger a Cancel Intermediate Event attached to the Sub-Process boundary. In addition, it will indicate that a Transaction Protocol Cancel message should be sent to any Entities involved in the Transaction.

Currently there are no attributes provided for this event.

#### 7.6.4.5.7. *TriggerResultSignal*

A Signal is for general communication within and across Process Levels, across Pools, and between Business Process Diagrams. A BPMN Signal is similar to a signal flare that shot into the sky for anyone who might be interested to notice and then react. Thus, there is a source of the Signal, but no specific intended target. This is different than a BPMN Message, which has a specific Source and a specific Target (which can be an Entity or an abstract Role).

For Start Event:

A signal arrives [CATCH] that has been broadcast from another Process and triggers the start of the Process. Note that the Signal is not a Message, which has a specific target for the Message. Multiple Processes can have Start Events that are triggered from the same broadcasted Signal.

For End Event:

This type of End indicates that a Signal will be broadcast [THROW] when the End has been reached. Note that the Signal, which is broadcast to any Process that can receive the Signal, can be sent across Process levels or Pools, but is not a Message (which has a specific Source and Target).

For Intermediate Event:

This type of Intermediate Event can send [THROW] or receive [CATCH] a Signal if the Event is part of a Normal Flow. The Event can only receive a Signal when attached to the boundary of an activity. The Signal Event differs from an Error Event in that the Signal defines a more general, non-error condition for interrupting activities (such as the successful completion of another activity) as well as having a larger scope than Error Events. When used to “catch” the signal, the Event marker will be unfilled. When used to “throw” the signal, the Event marker will be filled.

Schema in XSD file.

	Description
Name	Name is an attribute that is text description of the Signal.
Properties	Multiple Properties MAY be entered for the Signal

*Table : Trigger Result Signal*

#### 7.6.4.5.8. *TriggerIntermediateMultiple*

If the Trigger is a Multiple, then each Trigger on the list MUST have the appropriate data as specified for the attributes. The Trigger MUST NOT be of type None or Multiple.

Schema in XSD file.

	Description
	List of applicable triggers.TriggerResultMessage, TriggerTimer, ResultError, TriggerEscalation, TriggerResultCompensation, TriggerConditional, TriggerResultLink,TriggerResultCancel, TriggerResultSignal.

*Table : Event Trigger Intermediate Multiple*

#### 7.6.4.5.9. *TriggerMultiple*

If the Trigger attribute is a Multiple, then a list of two or more Triggers MUST be provided. Each Trigger MUST have the appropriate data. The Trigger MUST NOT be of type None or Multiple.

Schema in XSD file.

	Description
	List of applicable triggers: TriggerResultMessage,TriggerTimer, ResultError, TriggerEscalation, TriggerResultCompensation, TriggerConditional, TriggerResultSignal.

*Table : Event Trigger Multiple*



#### 7.6.4.5.10. *TriggerConditional*

If the Trigger is a Conditional, then a Condition **MUST** be entered.

For Intermediate Event:

This is only used for exception handling. This type of event is triggered when a Condition becomes true. A Condition is an expression that evaluates some Process data.

Schema in XSD file.

	Description
ConditionName	Name is an optional attribute that is a text description of the Condition. If a Name is not entered, then a ConditionExpression <b>MUST</b> be entered (see the attribute below).
Expression	A ConditionExpression may be entered. In some cases the Condition itself will be stored and maintained in a separate application (e.g., a Rules Engine). If a ConditionExpression is not entered, then a Name <b>MUST</b> be entered (see the attribute above).

*Table : Trigger Conditional*

#### 7.6.4.5.11. *TriggerTimer*

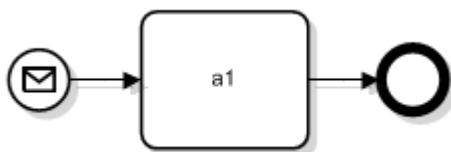
If the Trigger is a Timer, then a TimeDate **MAY** be entered. If a TimeDate is not entered, then a TimeCycle **MUST** be entered (see the elements below).

Schema in XSD file.

	Description
TimeDate	Expression. Should evaluate to one of the Basic Types: DATETIME, DATE or TIME. (See section 7.13.1).
TimeCycle	Expression. Should evaluate to an INTEGER representing the milliseconds duration of the cycle.

*Table : Event Trigger Timer*

#### 7.6.4.6. *Examples of Events and their representation*



```

<Activities>
  <Activity Id="3" Name="">
    <Event >
      <StartEvent Trigger="Message"></StartEvent>
    </Event>
  </Activity>
  <Activity Id="4" Name="a1"/>
  <Activity Id="5" Name="">
    <Event >
      <EndEvent Result="None"/>
    </Event>
  </Activity>
</Activities>
<Transitions>
  <Transition Id="6" Name="" From="3" To="4" FlowType="SequenceFlow"/>
  <Transition Id="7" Name="" From="4" To="5" FlowType="SequenceFlow"/>
</Transitions>
  
```

#### 7.6.4.7. *Review of Events*

Start Events, both Top-Level and Event Sub-Process, have ‘Triggers’ that define the cause for the event. There are multiple ways that these events can be triggered. Start Events can only react to (“catch”) a Trigger.

End Events may define a “Result” that is the consequence of a Sequence Flow ending. End Events can only create

(“throw”) a Result. Intermediate Boundary Events can only Catch a trigger. Some intermediate “in the sequence flow” Events can only Catch, some can only Throw and some can do both.

For the Events that catch, the markers are unfilled, and for Events that throw, the markers are filled.

In addition, there is a distinction between Interrupting and non-interrupting Events. This distinction applies to some Event Sub-Process Start Events and some Boundary Events. An interrupting Event terminates the Sub-Process or Task it is associated with, while a non-interrupting Event creates a separate “parallel” execution thread.

BPMN provides specific graphics for the event types supported:
































































Types	Start			Intermediate				End
	Top-Level	Event Sub-Process interrupting	Event Sub-Process non-interrupting	Catching	Boundary interrupting	Boundary non-interrupting	Throwing	
None								
Message								
Timer								
Error								
Escalation								
Cancel								
Compensation								
Conditional								
Link								
Signal								
Terminate								
Multiple								
Parallel Multiple								

Figure 7.: Event Subtypes – Catching and Throwing

### 7.6.5. Implementation Alternatives

It is assumed that the execution of the Activity is atomic with respect to the data under control of the Process or Workflow engine. That implies that in the case of a system crash, an abort, or a cancellation of the Activity, the

Relevant data field and the control data are rolled back (automatically or by other means), or an appropriate compensating activity is applied. (This does not necessarily hold for audit data.) This version of the specification does not include any specific controls over data synchronization or recovery (for example between execution, subflows or applications under execution).

Schema in XSD file.

An Activity may be implemented in one of six ways as described in the following table:

	Description
No implementation	Implementation by manual procedures (i.e. not supported by process or workflow).
Tool	Implementation is supported by (one or more) application(s). Deprecated, see Task.
Task	Implementation by a BPMN Task or Application (Tool). See section 7.6.5.3.
SubFlow/ ProcessRef	Implementation by another process. Note that BlockActivity is used for ActivitySet/Embedded subprocess. An ActivitySet with the TriggeredByEvent attribute indicates an Event Sub-Process.
Reference	Implementation by BPMN reference. See section 7.6.5.5. Deprecated in BPMN2.0.
GlobalActivityReference	Attribute GlobalActivityId references a BPMN2.0 globalTask defined in GlobalActivities. Pointer to Activity/@Id in XPDL.

*Table : Implementation Alternatives*

#### **7.6.5.1. No Implementation**

No Implementation means that the implementation of this Activity is not supported by Workflow using automatically invoked applications or procedures. Two Alternatives have been identified as to how this may be used:

It is a Manual Activity. In this case FinishMode value Manual is required.

It is an "implicit" activity, which is known to the Process Engine (e.g. by vendor-specific Extended Attributes) in terms of any processing requirements. An example is the Pre- and Post-processing Activities in a Process, which generate and clear hidden data when starting and terminating a process (e.g. managing the relationship to imaging system and archive). In this case the StartMode and FinishMode values Automatic are common.

Note that application initiation may still be handled directly by the participant under local control in a manual activity; this lies outside the scope of the specification.

Schema in XSD file.

#### **7.6.5.2. Tool**

Deprecated. See TaskApplication 7.6.5.3.10

#### **7.6.5.3. Task**

A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail. BPMN includes a set of built-in Tasks. Generally, an end-user and/or an application and/or WebService are used to perform the Task when it is executed. A Task object shares the same shape as the Sub-Process, which is a rectangle that has rounded corners.

##### **7.6.5.3.1. BPMN Graphics for Tasks**

A Task is a rounded corner rectangle that **MUST** be drawn with a single thin black line.

BPMN specifies three types of markers for Task: a Loop Marker or a Multiple Instance Marker and a Compensation Marker. A Task may have one or two of these markers

- The marker for a Task that is a standard loop **MUST** be a small line with an arrowhead that curls back upon itself.
  - The Loop Marker **MAY** be used in combination with the Compensation Marker.

- The marker for a Task that is a multi-instance loop **MUST** be a set of three vertical lines in parallel
  - The Multiple Instance Marker **MAY** be used in combination with the Compensation Marker
- The marker for a Task that is used for compensation **MUST** be a pair of left facing triangles (like a tape player “rewind” button).
  - The Compensation Marker **MAY** be used in combination with the Loop Marker or the Multiple Instance Marker.
- All the markers that are present **MUST** be grouped and the whole group centered at the bottom of the shape.

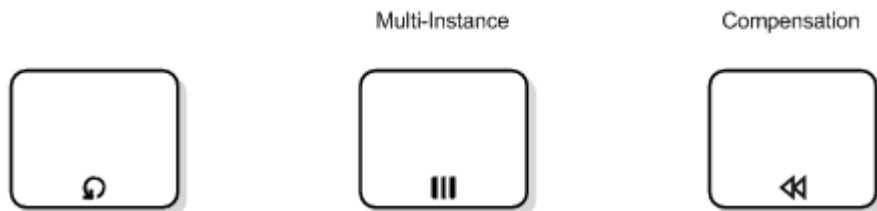


Figure 7.: Task Markers

In addition to categories of Task shown above, there are different types of Tasks identified within BPMN to separate the types of inherent behavior that Tasks might represent. However, BPMN1.x does not specify any graphical indicators for the different types of Tasks. Modelers or modeling tools may choose to create their own indicators or markers to show the readers of the diagram the type of Task. This is permitted by BPMN as long as the basic shape of the Task (a rounded rectangle) is not modified. The list of Task types may be extended along with any corresponding indicators.

BPMN2.0 provides specific graphical indicators which we illustrate for each task type.

#### 7.6.5.3.2. Schema for Tasks

The Activity is implemented by one of the eight BPMN tasks or by an Application (Tool).

Schema in XSDfile.

#### 7.6.5.3.3. TaskManual



A Manual Task is a Task that is expected to be performed without the aid of any business process execution engine or any application. An example of this could be a telephone technician installing a telephone at a customer location.

A TaskType of Manual **MUST NOT** have an incoming or an outgoing Message Flow.

Schema in XSD file.

	Description
Performers	A list of performers that will be performing the Manual Task. The Performers entry could be in the form of a specific individual, a group, or an organization. Similar to the 'Implementation:No' activity (see section 7.6.5.1).

Table : Task Manual

#### 7.6.5.3.4. TaskReceive



A TaskType of Receive MUST NOT have an outgoing Message Flow.

A Receive Task is a simple Task that is designed to wait for a message to arrive from an external participant (relative to the Business Process). Once the message has been received, the Task is completed. A Receive Task is often used to start a Process. In a sense, the Process is bootstrapped by the receipt of the message. In order for the Task to Instantiate the Process it must meet one of the following conditions:

- The Process does not have a Start Event and the Receive Task has no incoming Sequence Flow
- The Incoming Sequence Flow for the Receive Task has a source of a Start Event.
  - Note that no other incoming Sequence Flows are allowed for the Receive Task (in particular, a loop connection from a downstream object).

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified
Instantiate	Receive Tasks can be defined as the instantiation mechanism for the Process with the Instantiate attribute. This attribute MAY be set to true if the Task is the first activity after the Start Event or a starting Task if there is no Start Event. Multiple Tasks MAY have this attribute set to True.
Message changed to MessageRef BPMN1.1	A Message for the Message attribute Should be entered as an implementation detail. This indicates that the Message will be received by the Task. The Message in this context is equivalent to an in-only message pattern (Web service). One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task. See section 7.9.4.
WebServiceOperation	Describes the web service operation to be used by this task. (See section 7.9.6)

*Table : Task Receive*

#### 7.6.5.3.5. TaskReference

Deprecated in XPDL2.2. Now use GlobalActivityReference, a sub-element of Implementation.

There may be times where a modeler may want to reference another activity that has been defined. If the two (or more) activities share the exact same behavior, then by one referencing the other, the attributes that define the behavior only have to be created once and maintained in only one location.

Schema in XSD file.

	Description
TaskRef	The Task being referenced MUST be identified.

*Table : TaskReference*

#### 7.6.5.3.6. TaskSend



A TaskType of Send MUST NOT have an incoming Message Flow.

A Send Task is a simple Task that is designed to send a message to an external participant (relative to the Business Process). Once the message has been sent, the Task is completed.

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified
Message BPMN1.1 MessageRef	A Message for the Message attribute should be entered as implementation detail. This indicates that the Message will be sent by the Task. The Message in this context is equivalent to an out-only message pattern (Web service). One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task. See section 7.9.4
WebServiceOperation	Describes the web services operation to be used by this task. See section 7.9.6.
WebServiceFaultCatch	Describes how to process faults generated by the web service operation in this task. See section 7.9.7.

Table : TaskSend

#### 7.6.5.3.7. TaskService



A Service Task is a Task that provides some sort of service, which could be a Web service or an automated application.

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified
MessageIn Editors note: InMessageRef in BPMN1.1	A Message for the InMessage attribute may be entered. This indicates that the Message will be received at the start of the Task, after the availability of any defined InputSets (child element of Activity). One or more corresponding incoming Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all incoming Message Flow, but can arrive for only one of the incoming Message Flow for a single instance of the Task. See Section 7.9.4.
MessageOut Editors note: OutMessageRef in BPMN1.1	A Message for the OutMessage attribute may be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet (child element of Activity). One or more corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flow and the Message will be sent down all outgoing Message Flow at the completion of a single instance of the Task. See section 7.9.4.
WebServiceFaultCatch Editors note: Not in BPMN1.1	Describes how to process faults generated by the web service operation in this task. See section 7.9.7.
WebServiceOperation	Describes the web services operation to be used by this task. See section 7.9.6.

Table : Task Service

#### 7.6.5.3.8. TaskScript



A Task Type of Script MUST NOT have an incoming or an outgoing Message Flow.

A Script Task is executed by a business process engine. The modeler or implementer defines a script in a language that the engine can interpret. When the Task is ready to start, the engine will execute the script. When the script is completed, the Task will also be completed.

Schema in XSD file.

	Description
Script	The modeler MAY include a script that can be run when the Task is performed. If a script is not included, then the Task will act equivalent to a TaskType of None.

*Table : Task Script*

#### 7.6.5.3.9. TaskUser



A User Task is a typical “workflow” task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort.

Schema in XSD file.

	Description
Implementation	WebService   Other   Unspecified   HumanTaskWebService
MessageIn Editors note: changed to InMessageRef in BPMN1.1	A Message for the InMessage attribute may be entered. This indicates that the Message will be sent at the start of the Task, after the availability of any defined InputSets. A corresponding outgoing Message Flow MAY be shown on the diagram. However, the display of the Message Flow is not required. See section 7.9.4.
MessageOut Editors note: changed to OutMessageRef	A Message for the OutMessage attribute may be entered. The sending of this message marks the completion of the Task, which may cause the production of an OutputSet. One or more corresponding outgoing Message Flows MAY be shown on the diagram. However, the display of the Message Flow is not required. The Message is applied to all outgoing Message Flows and the Message will be sent down all outgoing Message Flows at the completion of a single instance of the Task. See section 7.9.4.
Performers	One or more Performers MAY be entered. The Performers attribute defines the human resource that will be performing the Task. The Performers entry could be in the form of a specific individual, a group, or an organization.
WebServiceOperation	Describes the web services operation to be used by this task. See section 7.9.6.

*Table : Task User*

#### 7.6.5.3.10. TaskApplication (Tool)

The Activity is implemented by (one or more) tools. A tool may be an application program (link to entity Application); which may be invoked via Interface 3 (WfMC) - see the Workflow Client Application API (WAPI - Interface 2). TaskApplication is deprecated in XPDL2.2.

Schema in XSD file.



	Description
Actual Parameters	A list of parameters to be passed to the subflow/subprocess. See section 7.1.6.1.
DataMappings	Alternative approach to passing values between process and application. See section 7.6.5.4.7.
Description	Textual description.
ExtendedAttributes	Optional extensions to meet individual implementation needs.
Id	Identifier used to identify the application or procedure, depending on the Type.
Name	Name used to identify the application or procedure.
PackageRef	Used if the application is not in this package.

Table : Tool

#### 7.6.5.3.11. Task Sequence Flow Connections

- A Task MAY be a target for Sequence Flow; it can have multiple incoming Flow. Incoming Flow MAY be from an alternative path and/or a parallel path.

**Note** – If the Task has multiple incoming Sequence Flows, then this is considered uncontrolled flow. This means that when a Token arrives from one of the Paths, the Task will be instantiated. It will not wait for the arrival of Tokens from the other paths. If another Token arrives from the same path or another path, then a separate instance of the Task will be created. **In effect, multiple incoming sequence flow is treated as if there was an implicit Exclusive Merge.** If the flow needs to be controlled, then the flow should converge with a Gateway that precedes the Task. **XPDL provides the concept of TransitionRestriction which is applicable to all activity types, including Task. Hence it is possible to model controlled flow without using a gateway. However, this may create problems in transferring a process definition to a BPMN editor.**

- If the Task does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Task MUST be instantiated when the process is instantiated.
  - Exceptions to this are Tasks that are defined as being Compensation activities (have the Compensation Marker). Compensation Tasks are not considered a part of the Normal Flow and MUST NOT be instantiated when the Process is instantiated.
- A Task MAY be a source for Sequence Flow; it can have multiple outgoing Flow. If there are multiple outgoing Sequence Flow, then this means that a separate parallel path is being created for each Flow. **In effect, multiple outgoing sequence flow is treated as if there was an implicit Parallel Gateway with diverging sequence flows.**

Tokens will be generated for each outgoing Sequence Flow from the Task. The TokenIds for each of the Tokens will be set such that it can be identified that the Tokens are all from the same parallel Fork as well as the number of Tokens that exist in parallel. **Editorial comment: This doesn't mention having conditions associated with each path. See 'conditional flow in section 7.7.2.2. Also note that XPDL transition restrictions support the more general case. In any case multiple outgoing sequence flows are represented in XPDL using TransitionRestriction Split elements with a list of references to the outgoing transitions (see section 7.6.9.2).**

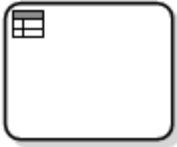
- If the Task does not have an outgoing Sequence Flow, and there is no End Event for the Process, then the Task marks the end of one or more paths in the Process. When the Task ends and there are no other parallel paths active, then the Process MUST be completed.
  - Exceptions to this are Tasks that are defined as being Compensation activities (have the Compensation Marker). Compensation Tasks are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

#### 7.6.5.3.12. Task Message Flow Connections

**Note** – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool.

- A Task *MAY* be the target for Message Flow; it can have zero or more incoming Message Flow. If there is multiple incoming Message Flow, then a single Message will be applied to all the Message Flow. However, only one Message can be received, from a single Message Flow, for a given instance of the Task
- A Task *MAY* be a source for Message Flow; it can have zero or more outgoing Message Flow. If there are multiple outgoing Message Flows, then a single Message will be applied to all the Message Flow. That Message will be sent down all the outgoing Message Flows.

#### 7.6.5.3.13. TaskBusinessRule



A Business Rule Task provides a mechanism for the Process to provide input to a Business Rules Engine and to get the output of calculations that the Business Rules Engine might provide. The InputOutputSpecification of the Task will allow the Process to send data to and receive data from the Business Rules Engine.

	Description
BusinessRuleTaskImplementation	BusinessRuleWebService WebService   Other   Unspecified

#### 7.6.5.4. SubFlow/Sub-Process

The Activity is refined as a subflow. The subflow may be executed synchronously or asynchronously. The subflow identifiers used are inherited from the surrounding Package declaration.

SubFlow calls are transactional if the parent Activity has been defined that way. See Table .

In the case of **asynchronous execution** the execution of the Activity is continued after a process instance of the referenced Process Definition is initiated (in this case execution proceeds to any post activity split logic after subflow initiation. No return parameters are supported from such called processes. Synchronization with the initiated subflow, if required, has to be done by other means such as events, not described in this document. This style of subflow is characterized as chained (or forked) subflow operation.

In the case of **synchronous execution** the execution of the Activity is suspended after a process instance of the referenced Process Definition is initiated. After execution termination of this process instance the Activity is resumed. Return parameters may be used between the called and calling processes on completion of the subflow. This style of subflow is characterized as hierarchic subflow operation.

##### 7.6.5.4.1. Remote SubProcess Node

A Subflow activity is a node in a process which invokes another process, typically passing formal parameters to it, and receiving result values back.

The ASAP and Wf-XML protocols allow for the same sort of interaction with a subprocess which is located remotely, and accessed purely through SOAP calls. The parent process runs on one server, and the subprocess on a different server, possibly implemented with different technology.

This is an extension to the SubFlow, to allow the specification of the "End Point Reference" of the remote web service that represents the factory of the subprocess. End Point Reference is defined by WS-Addressing, and most commonly appears as a URL but is not necessarily limited to this.

A workflow/BPM engine that understands how to perform a remote subprocess interaction will offer in the design tool the ability for the user to specify the URI of the remote process (factory). This URI might be found in any number of ways including being listed on a web page, entered as a UDDI entry, or retrieved using a Wf-XML ListFactories request. The data mapping then is set up in the same way as for a normal subprocess. When this system writes out the XPD, it will include this new tag in the SubFlow tag. The presence of this tag will indicate that this is a remote subprocess; the omission of it indicates that this is not a remote subprocess.

#### 7.6.5.4.2. Schema for SubFlow/Sub-Process

Schema in XSD file.

	Description
Actual Parameters	A list of parameters to be passed to the subflow. See section 7.1.6.1.
DataMapping	Alternative approach to passing values between caller and called process. See section 7.6.5.4.7.
PackageRef	Used if the subflow/process is not this package.
EndPointRef	To handle remote subprocess node.
Execution	<div>ASYNCHR Executed asynchronously.</div> <div>SYNCHR Executed synchronously. Is default.</div>
ExtendedAttributes	See section 6.4.14.
Id	Used to identify the process that is invoked.
InstanceDataField	The name of the DataField in which to store the subflow instance id for subsequent use such as messaging or correlation. Typically used with ASYNCHR execution.
Name	The name of the SubFlow/Process.
StartActivityId	If present, StartActivityId must be the id of a start activity: <ul style="list-style-type: none"> <li>In the referenced Activity Set if that's present</li> <li>In the top level activities of the referenced process otherwise</li> </ul>
StartActivitySetId	If present, StartActivitySetId must match the id of an activity set in the referenced process.
View	Indicates whether the activity is COLLAPSED or EXPANDED.

Table : SubFlow

BPMN provides a specific graphical representation for the different activities: The BPMN CallActivity, when referencing a Global Process, is mapped to an XPDL Sub-Process. (Note that BPMN Embedded Sub-Process Activity is mapped to an XPDL BlockActivity pointing at an ActivitySet).

The SubFlow [Sub-Process] can be in a collapsed view that hides its details or in an expanded view that shows its details within the view of the Process in which it is contained. In the collapsed form, the Sub-Process activity uses a marker to distinguish it as a Sub-Process, rather than a Task or other simple activity. Note the thick border which in BPMN distinguishes it as a re-usable sub-process rather than an embedded sub-process.

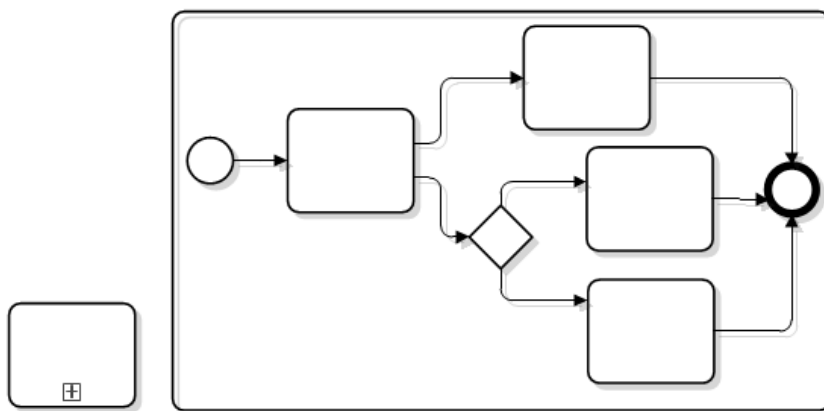


Figure 7.: Collapsed and Expanded Subflow

Expanded Sub-Processes may be used for multiple purposes. They can be used to “flatten” a hierarchical process so that all detail can be shown at the same time. They are used to create a context for exception handling that applies to a group of activities. Compensations can be handled similarly.

Some BPMN editors may not support the EXPANDED VIEW.

Expanded Sub-Process may be used as a mechanism for showing a group of parallel activities in a less-cluttered, more compact way. In the figure below, activities “C” and “D” are enclosed in an unlabeled Expanded Sub-Process. These two activities will be performed in parallel. Notice that the Expanded Sub-Process does not include a Start Event or an End Event and the Sequence Flow to/from these Events. This usage of Expanded Sub-Processes for “parallel boxes” is the motivation for having Start and End Events being optional objects.

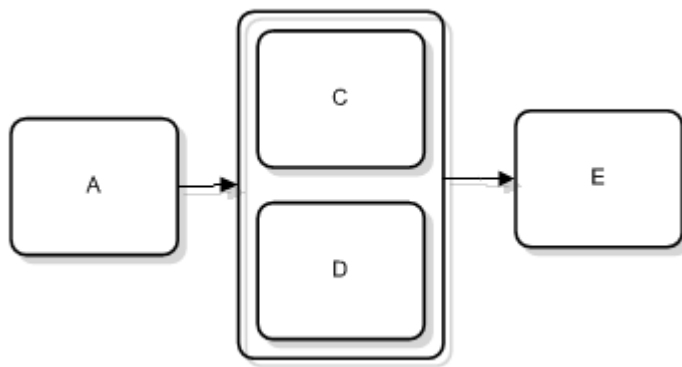


Figure 7.: Expanded Subprocess for parallel activities

BPMN specifies five types of standard markers for Sub-Processes. The (Collapsed) Sub-Process Marker can be combined with four other markers: a Loop Marker or a Parallel Marker, a Compensation Marker, and an Ad Hoc Marker. A collapsed Sub-Process may have one to three of these other markers, in all combinations except that Loop and Multiple Instance cannot be shown at the same time

- The marker for a Sub-Process that loops MUST be a small line with an arrowhead that curls back upon itself.
  - The Loop Marker MAY be used in combination with any of the other markers except the Multiple Instance Marker.
- The marker for a Sub-Process that has multiple instances MUST be a set of three vertical lines in parallel.
  - The Multiple Instance Marker MAY be used in combination with any of the other markers except the Loop Marker.
- The marker for a Sub-Process that is Ad Hoc MUST be a “tilde” symbol.
  - The Ad-Hoc Marker MAY be used in combination with any of the other markers
- The marker for a Sub-Process that is used for compensation MUST be a pair of left facing triangles (like a tape player “rewind” button).
  - The Compensation Marker MAY be used in combination with any of the other markers.
- All the markers that are present MUST be grouped and the whole group centered at the bottom of the shape.



Figure 7.: SubProcess Markers

#### 7.6.5.4.3. Semantics of ReusableSubprocess [BPMN perspective]

A Reusable Sub-Process object is an activity within a Process that “calls” to another Process. The Process that is called is not dependent on the Reusable Sub-Process object’s parent Process for global data. The Reusable Sub-Process object may pass data to/from the called Process [see DataMapping].

The called Process will be instantiated as a Sub-Process through a ‘None’ Start Event. Being reusable, the Process could also be instantiated as a Sub-Process by other Independent Sub- Process objects (in the same or other diagrams). In addition, it can be instantiated as a top-level Process through a separate Start Event that has a Trigger (other than None)

[See diagram below].

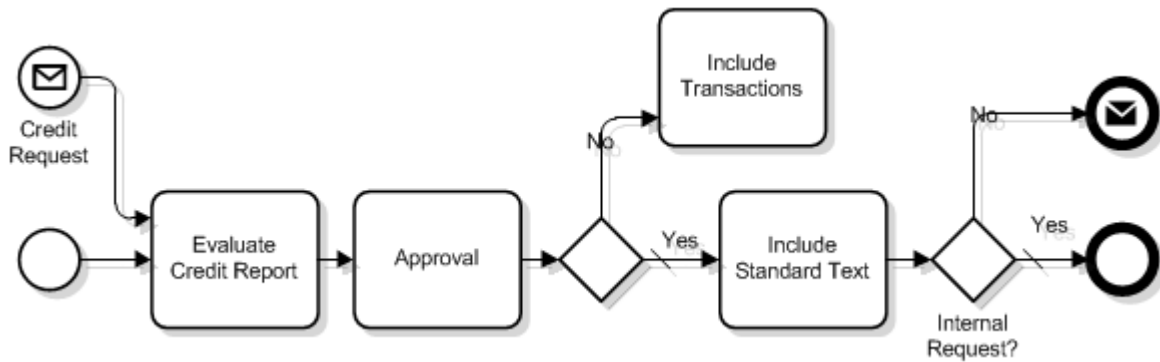


Figure 7.: Reusable Subprocess Semantics: BPMN perspective

#### 7.6.5.4.4. **Sub-Process Behavior as a Transaction**

A Sub-Process, either collapsed or expanded, can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol. The boundary of the activity will be double-lined to indicate that it is a Transaction (see Figure below).

Refer to section 7.6.12 Transaction for details.

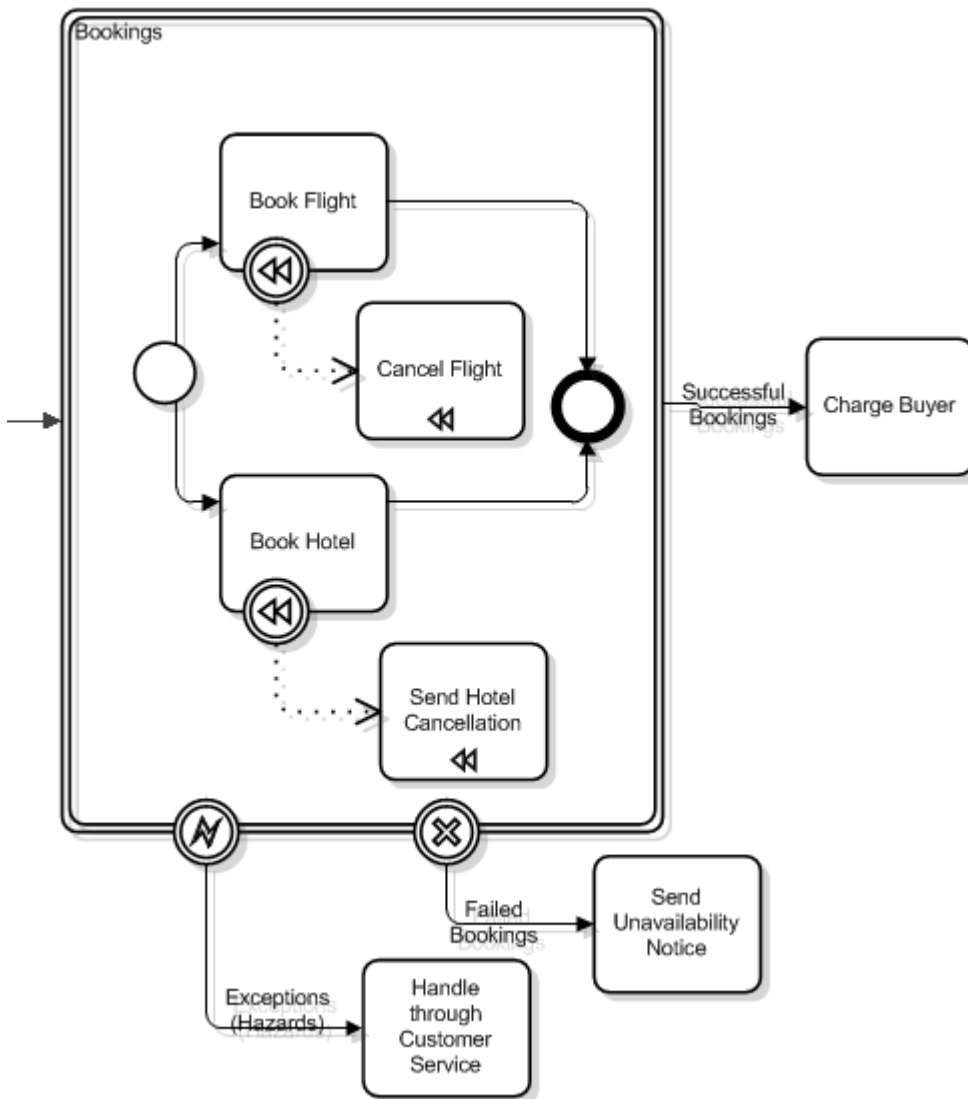


Figure 7.: Subprocess behavior as a Transaction

There are three basic outcomes of a Transaction:

- Successful completion: this will be shown as a normal Sequence Flow that leaves the Sub-Process.
- Failed completion (Cancel): When a Transaction is cancelled, then the activities inside the Transaction will be subjected to the cancellation actions, which could include rolling back the process and compensation for specific activities. Note that other mechanisms for interrupting a Sub-Process will not cause Compensation (e.g., Error, Timer, and anything for a non-Transaction activity). A Cancel Intermediate Event, attached to the boundary of the activity, will direct the flow after the Transaction has been rolled back and all compensation has been completed. The Cancel Intermediate Event can only be used when attached to the boundary of a Transaction activity. It cannot be used in any Normal Flow and cannot be attached to a non-Transaction activity. There are two mechanisms that can signal the cancellation of a Transaction:
  - A Cancel End Event is reached within the Transaction Sub-Process. A Cancel End Event can only be used within a Sub-Process that is set to a Transaction.
  - A Cancel Message can be received via the Transaction Protocol that is supporting the execution of the Sub- Process.
- Hazard: This means that something went terribly wrong and that a normal success or cancel is not possible. We are using an Error to show Hazards. When a Hazard happens, the activity is interrupted (without Compensation) and the flow will continue from the Error Intermediate Event.

The behavior at the end of a successful Transaction Sub-Process is slightly different than that of a normal Sub-Process.

When each path of the Transaction Sub-Process reaches a non-Cancel End Event(s), the flow does not immediately move back up to the higher-level Parent Process, as does a normal Sub-Process. First, the transaction protocol must verify that all the participants have successfully completed their end of the Transaction. Most of the time this will be true and the flow will then move up to the higher-level Process. But it is possible that one of the participants may end up with a problem that causes a Cancel or a Hazard. In this case, the flow will then move to the appropriate Intermediate Event, even though it had apparently finished successfully

#### 7.6.5.4.5. SubProcess Activity Sequence Flow Connections

- A Sub-Process MAY be a target for Sequence Flow; it can have multiple incoming Flow. Incoming Flow MAY be from an alternative path and/or a parallel paths.
  - The Incoming Sequence Flow's attribute TargetRef MAY be extended to include both the Sub-Process object (at the parent level) and a Start Event that resides within the details of the Sub-Process. This provides a direct connection from the parent-level Sequence Flow to the lower-level Start Event for situations where there is more than one Start Event in the Sub-Process. The form of the extension would be "Sub-Process.Start".
    - Note that XPDL provides an alternate method for specifying which start event: the attribute StartActivityID of the Subflow activity
    - If the details of the Sub-Process (i.e., its Start Events) are not visible or accessible to the modeler, then the determination as to which Start Event, if there are multiple, will be triggered is undefined. But only one of the Start Events will be triggered.

**Note** – If the Sub-Process has multiple incoming Sequence Flows, then this is considered uncontrolled flow. This means that when a Token arrives from one of the Paths, the Sub-Process will be instantiated. It will not wait for the arrival of Tokens from the other paths. If another Token arrives from the same path or another path, then a separate instance of the Sub-Process will be created. **In effect, multiple incoming sequence flow is treated as if there was an implicit Exclusive Merge.** If the flow needs to be controlled, then the flow should converge on a Gateway that precedes the Sub-Process. **XPDL provides the concept of TransitionRestriction which is applicable to all activity types, including Subflow/Subprocess. Hence it is possible to model controlled flow without using a gateway. However, this may create problems in transferring a process definition to a BPMN editor.**

- If the Sub-Process does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Sub-Process MUST be instantiated when the process is instantiated.
  - Exceptions to this are Sub-Processes that are defined as being Compensation activities (have the Compensation Marker). Compensation Sub-Processes are not considered a part of the Normal Flow and MUST NOT be instantiated when the Process is instantiated.
- A Sub-Process MAY be a source for Sequence Flow; it can have multiple outgoing Flow. If there are multiple outgoing Sequence Flow, then this means that a separate parallel path is being created for each Flow. **In effect, multiple outgoing sequence flow is treated as if there was an implicit Parallel Gateway with diverging sequence flows.**

Tokens will be generated for each outgoing Sequence Flow from Sub-Process. The TokenIds for each of the Tokens will be set such that it can be identified that the Tokens are all from the same parallel Fork as well as the number of Tokens that exist in parallel. **This doesn't mention having conditions associated with each path. See 'conditional flow' in section 7.7.2.2. Also note that XPDL transition restrictions support the more general case. In any case multiple outgoing sequence flows are represented in XPDL using TransitionRestriction Split elements with a list of references to the outgoing transitions (see section 7.6.9.2).**

- If the Sub-Process does not have an outgoing Sequence Flow, and there is no End Event for the Process, then the Sub-Process marks the end of one or more paths in the Process. When the Sub-Process ends and there are no other parallel paths active, then the Process MUST be completed.
  - Exceptions to this are Sub-Processes that are defined as being Compensation activities (have the Compensation Marker). Compensation Sub-Processes are not considered a part of the Normal Flow and MUST NOT mark the end of the Process.

#### 7.6.5.4.6. SubProcess Activity Message Flow Connections

**Note** – All Message Flow must connect two separate Pools. They can connect to the Pool boundary or to Flow Objects within the Pool boundary. They cannot connect two objects within the same Pool

- A Sub-Process MAY be the target for Message Flow; it can have zero or more incoming Message Flows.
- A Sub-Process MAY be a source for Message Flow; it can have zero or more outgoing Message Flows.

#### 7.6.5.4.7. *DataMapping*

```

<xsd:element name="DataMapping">
  <xsd:annotation>
    <xsd:documentation>XPDL and BPMN:Maps fields between calling and called processes or
subprocesses</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Actual" type="xpd:ExpressionType"/>
      <xsd:element name="TestValue" type="xpd:ExpressionType" minOccurs="0"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="Formal" type="xsd:string" use="required"/>
    <xsd:attribute name="Direction" default="IN"/>
    <xsd:simpleType>
      <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="IN"/>
        <xsd:enumeration value="OUT"/>
        <xsd:enumeration value="INOUT"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:complexType>
</xsd:element>

<xsd:element name="DataMappings">
  <xsd:annotation>
    <xsd:documentation>XPDL and BPMN:Maps fields or properties between calling and called processes or
subprocesses</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xpd:DataMapping" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

```

	Description
Actual	Name of DataField whose value should be passed. If Direction is 'IN' can be an expression.
Direction	IN   OUT   INOUT
Formal	Name of DataField in receiving subprocess or application.
TestValue	The purpose of this tag is to make it possible to enter test run or simulation values. This would enable the process engine to fake the call to the subprocess/application. Instead of making the call to application/sub process, an engine could just pass the specified test values as a return to the parent process, as if the call has really happened. The TestValue tag would be allowed only for the OUT and INOUT directed mappings.

Table : *DataMapping*

#### An Example showing use of TestValue



```

<DataMappings>
.....
  <DataMapping Formal="subprocParam1" Direction="OUT">
    <Actual>ParProcFieldName</Actual>
    <TestValue>This subprocess has not been invoked</TestValue>
  </DataMapping>
.....
</DataMappings>

```

#### 7.6.5.5. Reference

Deprecated in XPD2.2, see section 7.6.5.6 GlobalActivityReference instead.

There may be times where a modeler may want to reference another Sub-Process that has been defined. If the two SubProcesses share the exact same behavior and properties, then by one referencing the other, the attributes that define the behavior only have to be created once and maintained in only one location.

There may be times where a modeler may want to reference another activity that has been defined. If the two (or more) activities share the exact same behavior, then by one referencing the other, the attributes that define the behavior only have to be created once and maintained in only one location.

```

<xsd:element name="Reference">
  <xsd:annotation>
    <xsd:documentation>BPMN</xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="ActivityId" type="xpd:IdRef" use="required">
      <xsd:annotation>
        <xsd:documentation>BPMN: Reference to a BPMN task or subprocess definition elsewhere; should not be
used for gateway or event. Pointer to Activity/@Id in XPD2.</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>

```

	Description
Activity Id	The Id of the activity that defines the desired behaviour.

Table : Reference

#### 7.6.5.6. GlobalActivityReference

The GlobalActivityReference is used to represent the BPMN CallActivity when it refers to a GlobalTask. The marker for this is the same as the marker for the kind of Global Task referred to, except that the boundary is drawn with a thick line. For instance, if calling a Global User Task it would appear as follows:



It has a single attribute, GlobalActivityId which is an Idref of the CalledActivity in the list of Global Activities.

### 7.6.6. Performer Relationship

The relationship of the Activity to a (potential) performer is given by the Performers attribute (see section 6.4.5). It provides a link to the entity Participant. Default: Any Participant.

The Participant identifiers used in the Performer attribute have either to be declared in the surrounding Process definition or are inherited from the surrounding Package declaration or coming from external packages.

The question whether the expression evaluation results in an empty set of performers or a non unique performer is to be handled by the process or workflow management system at run time or, where defined, by the external resource repository or organizational model. The runtime resolution of both cases is outside the scope of this specification.

- In the first case (empty set) the engine may e.g. retry at a later time, or it may signal this to the supervisor of the process. The approach used is local to the WFMS and does not form part of this specification.
- The second case (non-unique) may arise where the performer definition is by function/skill type (defined as “Role”) and/or is an organization unit, which is itself a container for a set of participants. In these situations the approach adopted for Performer/participant assignment is local to the WFMS and does not form part of this specification. Common scenarios are:
  - Where an activity includes multiple work items that may be implemented in parallel, separate work items may be presented to a number of performers.
  - In other situations the activity may be assigned according to a local load-balancing algorithm or presented to multiple potential performers in their work lists and assigned to the first accepting participant. (It is the responsibility of the engine to provide the appropriate behavior.)
  - The assignment of an activity to an organizational unit (e.g. a department) may result in the activity being offered to all members of the organizational unit and assigned to the first accepting participant or allow the manager of the unit to redirect the activity to a designated departmental member.

In all cases the Performers/participant assignments defined within the meta-model and expressed in XPDL only relate Activities to defined Participants (including the use of expressions and defined Functions) and do not differentiate between cases where the defined Participant is atomic (e.g. a person) or not (e.g. a team). The local behavior of the engine and the resource repository or organizational model in handling these situations is not defined.

### 7.6.7. Deadline

Deadlines are used to raise an exception upon the expiration of a specific period of time.

Note that BPMN provides the Timer Event as a general method for handling Deadlines. See section 7.6.4.5.11.

Upon the arrival of a deadline, an exception condition is raised and the appropriate exception transitions are followed. If the deadline is synchronous, then the activity is terminated before flow continues on the exception path. If the deadline is asynchronous, then an implicit AND SPLIT is performed, and a new thread of processing is started on the appropriate exception transition. Asynchronous exceptions can cause side effects, and should be used carefully. Some of these side effects are discussed later in this section.

A sample deadline is below. In the sample, an asynchronous “notifyException” will be raised after 3 days. The activity will continue normally at this point. If the activity is still executing after 5 days it will be terminated and a “timeoutException” will be raised.

Sample Deadline

```
<Deadline Execution="ASYNCHR">
  <DeadlineDuration>3 days</DeadlineDuration>
  <ExceptionName>notifyException</ExceptionName>
</Deadline>
<Deadline Execution="SYNCHR">
  <DeadlineDuration>5 days</DeadlineDuration>
  <ExceptionName>timeoutException</ExceptionName>
</Deadline>
```

The syntax of the deadline duration is implementation dependent. The duration may be relative or absolute and may use relevant data field.

If a synchronous deadline occurs on a block activity or a subflow, stopping the activity includes stopping all the threads in the block, or the subflow and all its threads and nested subflows as well. From a modeling perspective, this can be treated as “immediate termination.” If an engine chooses to deviate from this, such as allowing an in-process manual activity to complete, it should document this behavior.

An asynchronous exception can be a powerful tool, allowing intermediate notification and graceful process termination by altering relevant data field. But an asynchronous exception can also create race conditions and possible side effects. For instance, the running activity could complete while the asynchronous exception is being processed. In addition,

because an implicit split is performed, flow control can be complicated if the asynchronous exception processing joins back up with the deadline processing thread. Care must be taken by the designer to properly handle race conditions and avoid unwanted side effects.

#### 7.6.7.1. Schema for Deadline

Schema in XSD file.

	Description
Execution	<p>Define the system behaviour on raising the arrival of the deadline</p> <p>ASYNCHR                      The deadline is to be raised asynchronously. This is an implicit AND SPLIT operation, where the activity continues and another thread is started following the named exception transition. Another deadline may occur on the same activity, because it continues running.</p> <p>SYNCHR                      The activity is completed abnormally and flow continues on the named exception transition.</p>
DeadlineDuration	<p>An expression indicating the time of the deadline. This expression is implementation dependent and may include at least:</p> <p>Times relative to the beginning of the activity. (2 days)</p> <p>Fixed times (January 1) or (January 1, 2010)</p> <p>Times computed using relevant data field (varName days)</p>
ExceptionName	The name of the exception to be raised on arrival of the deadline.

Table : Deadline

#### 7.6.8. Simulation Information

Schema in XSD file.

	Description
Cost	Average cost.
CostStructure	Adding detailed cost structure to simulation allows for greater comparison with real time results gathered from business activity monitoring tools as well as capturing greater detail typically housed in process engineering tools.
Duration	Expected duration time to perform a task in units of DurationUnit.
Instantiation	<p>Defines the capability of an activity to be activated. Defines how many times an Activity can be activated for higher throughput (e.g. how many individuals can capture a role). This can be once or many times (multiple).</p> <p>ONCE                      The Activity can only be instantiated once. Default.</p> <p>MULTIPLE                      The Activity can be instantiated multiple times.</p>
Time Estimation	Expected duration (summary of working time, waiting time, and duration) in units of DurationUnit.
Waiting Time	Average waiting time in units of DurationUnit.
Working Time	Average working time in units of DurationUnit.

*Table : Simulation Information*

### 7.6.9. Transition Restriction

**Any activity (including route activities) with multiple outgoing transitions (sequence flow) must have a SPLIT transition restriction with a list of references to the outgoing transitions.** The split transition restriction is used to specify the order in which the outgoing transitions are evaluated.

A JOIN transition restriction MAY be used when there are multiple incoming sequence flows, however in BPMN diagrams the join and split behavior for non-gateway activities is fixed (join is always exclusive and split is always inclusive).

Schema in XSD file.

	Description
Join	Specifies that the incoming Transitions of the Activity are JOIN-ed.
Split	Specifies that the outgoing Transitions of the Activity are SPLIT-ed.

*Table : Transition Restrictions*

#### 7.6.9.1. Join

A join describes the semantics of an activity with multiple incoming Transitions.

Note that for BPMN diagrams this element is no longer necessary because for gateway activities the type is specified in the Route element and for non-gateway activities the join behaviour is fixed as exclusive.

Schema in XSD file.

	Description
Type	<p>Specifies the join behavior of this activity. If specified, then for BPMN gateway activities this MUST match the GatewayType of the Route element. If specified for BPMN non-gateway activities then this MUST be “Exclusive”.</p> <p>Note that the enumerations XOR, OR and AND are deprecated and replaced by the new enumerations Exclusive, Inclusive and Parallel respectively. It is recommended that modellers are capable of reading deprecated values but always write the new enumerations.</p> <p>Parallel (AND) Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s). Equivalent to BPMN ‘OR’ gateway merge logic. BPMN ‘AND’ gateway merge logic requires all incoming transition threads to be synchronized, regardless of previous splits.</p> <p>Inclusive (OR) See above.</p> <p>Exclusive (XOR) Join for alternative threads: No synchronisation is required.</p> <p>COMPLEX This makes use of the attribute IncomingCondition (see below).</p>
ExclusiveType	Deprecated – join behaviour is identical for Exclusive Data-Base and Exclusive Event-based gateways.
IncomingCondition	An expression that determines which of the incoming transitions/Sequence Flow are required for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flow. For example, an expression may specify that any 3 out of 5 incoming Tokens will continue the Process. Another example would be an expression that specifies that a Token is required from Sequence Flow “a” and that a Token from either Sequence Flow “b” or “c” is acceptable. However, the expression should be designed so that the Process is not stalled at that location.

Table : Join

The Parallel (AND) join can be seen as a "rendezvous precondition" of the Activity; the activity is not initiated until the transition conditions on all incoming routes evaluate true.

The Exclusive (XOR) join initiates the Activity when the transition conditions of any (one) of the incoming transitions evaluates true.

### 7.6.9.2. Split

A split describes the semantics where multiple outgoing Transitions for an Activity exist. **Any activity (including route activities) with multiple outgoing transitions (sequence flow) must have a SPLIT transition restriction with a list of references to the outgoing transitions.** The split transition restriction is used to specify the order in which the outgoing transitions are evaluated.

Schema in XSD file.

	Description
Transition Refs	A list of outgoing transitions from the Activity. Each transition is identified by its Id. Mandatory for all activities (including gateways) that have multiple outgoing sequence flow.
Type	<p>Specifies the split behavior of this activity. If specified, then for BPMN gateway activities then this MUST match the GatewayType of the Route element. If specified for BPMN non-gateway activities then this MUST be "Inclusive".</p> <p>Note that the enumerations XOR, OR and AND are deprecated and replaced by the new enumerations Exclusive, Inclusive and Parallel respectively. It is recommended that modellers are capable of reading deprecated values but always write the new enumerations.</p> <p>Parallel (AND) Defines a number of possible concurrent threads represented by the outgoing Transitions of this Activity. If the Transitions have conditions the actual number of executed par-allel threads is dependent on the conditions associated with each transition, which are evaluated concurrently. Note that the BPMN <code>_AND</code> gateway is a special case where none of the transitions (sequence flow gates) have conditions.</p> <p>Inclusive (OR) Another BPMN variant of AND. Here the transitions may have conditions and multiple paths out may be chosen. The difference is that a default path may also be specified so that, in case none of the other paths are chosen, the default will be selected.</p> <p>Exclusive (XOR) with ExclusiveType "Data"</p> <p>List of Identifiers of outgoing Transitions of this Activity, representing. alternatively executed transitions. The decision as to which single transition route is selected is dependent on the conditions of each individual transition as they are evaluated in the sequence specified in the list. The first TRUE condition (or no condition) ends the list evaluation. An OTHERWISE transition should be chosen if no prior transition is chosen. Equivalent to BPMN XOR data based Gateway.</p> <p>Exclusive (XOR) with ExclusiveType "Event"</p> <p>Similar to XOR except that none of the transitions have conditions and there can be no OTHERWISE transition. The target activities must be Tasks with the TaskType attribute set to Receive or Intermediate Events with the Trigger attribute set to Message, Timer, Rule, or Link. If one transition/Gate target is a Task, then an Intermediate Event with a Trigger Message MUST NOT be used as a target for another transition/Gate. That is, messages MUST be received by only Receive Tasks or only Message Events, but not a mixture of both for a given Gateway.</p> <p>COMPLEX This makes use of the attribute OutgoingCondition (see below).</p>
ExclusiveType	Where the Type is "Exclusive" this attribute specifies whether it is an Exclusive Data-Based split or and Exclusive Event-Based split (see above).

	Description
OutgoingCondition	An expression determines which of the outgoing transitions/Sequence Flow will be chosen for the Process to continue. The expression may refer to process data and the status of the incoming Sequence Flow. For example, an expression may evaluate Process data and then select different sets of outgoing Sequence Flow, based on the results of the evaluation. However, The expression should be designed so that at least one of the outgoing Sequence Flow will be chosen.

*Table : Split*

A Parallel (AND) split with transitions having conditions may be referred to as "conditional AND", "multiple-choice OR", or "nonexclusive OR", respectively. The number of actual concurrent threads is determined at execution time when evaluating the conditions. Following such an AND split the process instance (or thread of the process instance) is forked into a number of separate execution threads which result from the transitions condition evaluation. (Note that no list of identifiers is required since all outgoing transitions from the activity are evaluated and no sequence is necessary.)

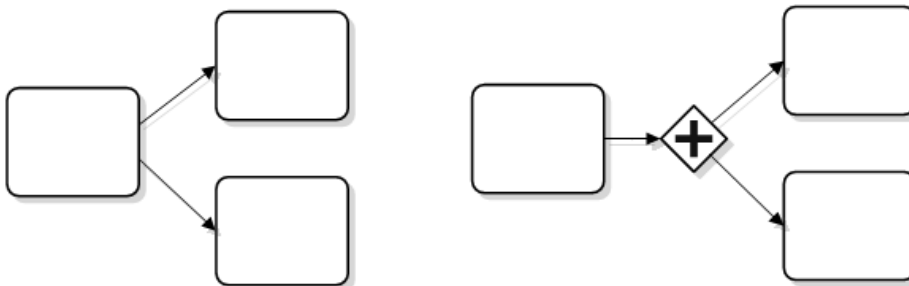
If within the AND\_SPLIT or XOR\_SPLIT there is a transition having condition OTHERWISE, then a two-step evaluation is performed. In the first step evaluation is made of all the Transitions except that within the OTHERWISE condition. If none of them (including those having no condition) evaluate to TRUE, then in the second step the OTHERWISE transition is evaluated (only one transition with an OTHERWISE clause is permitted in the list of outgoing transitions from an activity).

An OTHERWISE alternative can be used to guarantee that there is no undefined status from the Process execution (i.e. at least one outgoing transition from an activity will always occur).

### 7.6.9.3. BPMN View of Routing Logic

#### 7.6.9.3.1. Fork

BPMN uses the term “fork” to refer to the dividing of a path into two or more parallel paths (also known as an AND-Split **described in sections 7.6.2 and 7.6.9.2**). It is a place in the Process where activities can be performed concurrently, rather than sequentially. There are two options: Multiple Outgoing Sequence Flow can be used. This represents “uncontrolled” flow and is the preferred method for most situations. A Parallel Gateway can be used. This will be used rarely, usually in combination with other Gateways.

*Figure 7.: BPMN Fork*

#### 7.6.9.3.2. Join

BPMN uses the term “join” to refer to the combining of two or more parallel paths into one path (also known as an AND-Join or synchronization). A Parallel Gateway is used to show the joining of multiple Flow.

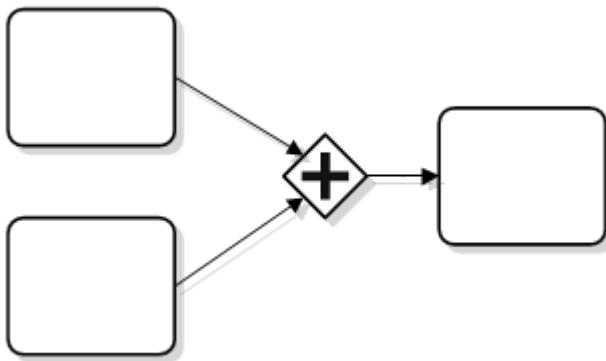


Figure 7.: BPMN Join

#### 7.6.9.3.3. Exclusive Data-Based Decision

This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow (“Data- Based” see section 7.6.2.2.1). Only one of the Alternatives will be chosen.

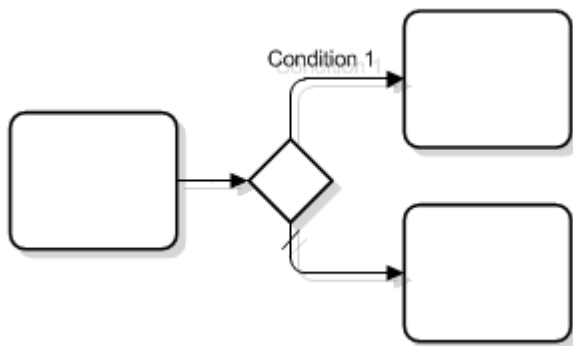


Figure 7.: BPMN Exclusive Data-Based Decision

#### 7.6.9.3.4. Exclusive Event-Based Decision

This Decision represents a branching point where Alternatives are based on an Event that occurs at that point in the Process (see section 7.6.4.3). The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the Alternatives will be chosen. There are two options for receiving Messages: Tasks of Type Receive can be used (below left). Intermediate Events of Type Message can be used (below right).

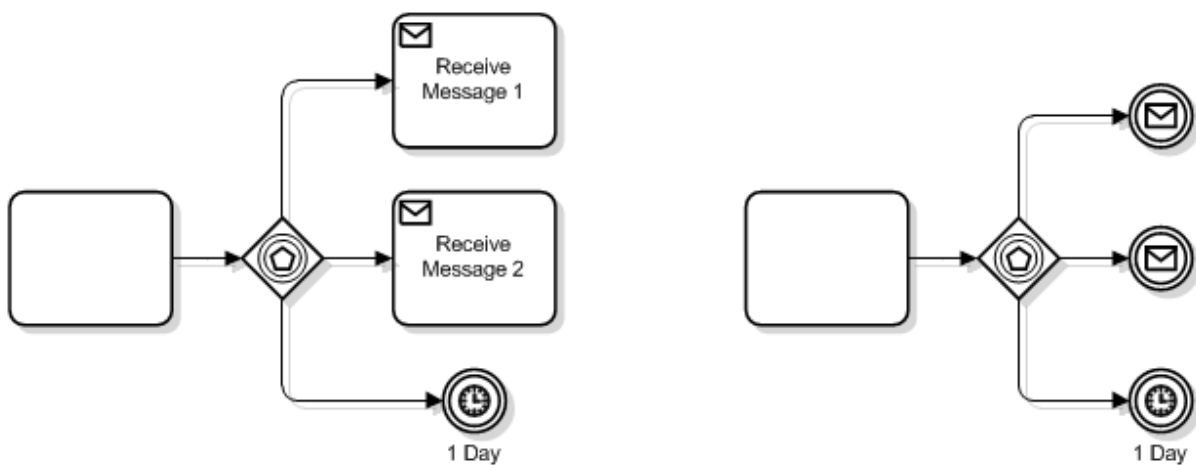


Figure 7.: BPMN Event-Based Decision

#### 7.6.9.3.5. Inclusive Decision

This Decision represents a branching point where Alternatives are based on conditional expressions contained within the outgoing Sequence Flow (7.6.2.1, 7.6.9.2). In some sense it is a grouping of related independent Binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. A Default Condition could be used to ensure that at least one path is taken. There are two versions of this type of Decision: The first uses a collection of conditional Sequence Flow, marked with mini-diamonds (see below left figure). The second uses a Gateway (see below right figure).

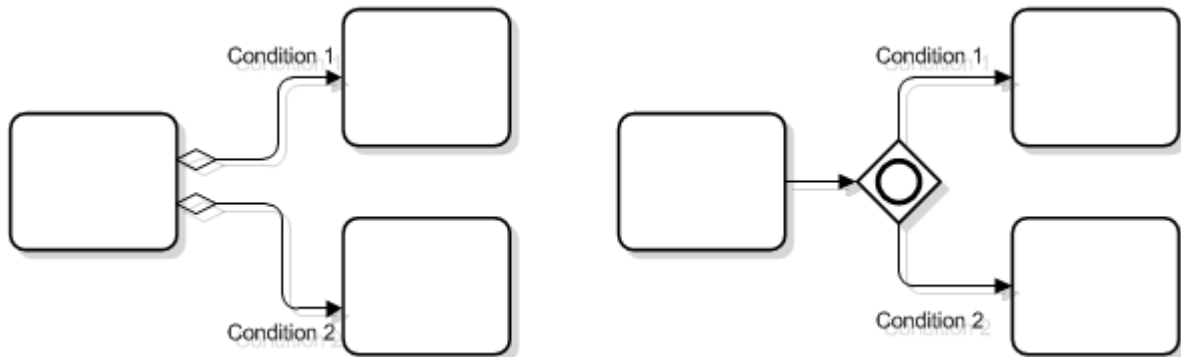


Figure 7.: BPMN Inclusive Decision

#### 7.6.9.3.6. Merging

BPMN uses the term “merge” to refer to the exclusive combining of two or more paths into one path (also known as an OR-Join described in section 7.6.2.2.2). A Merging Exclusive Gateway is used to show the merging of multiple Flow. If all the incoming flows are alternatives, then a Gateway is not needed. That is, uncontrolled flow provides the same behavior.

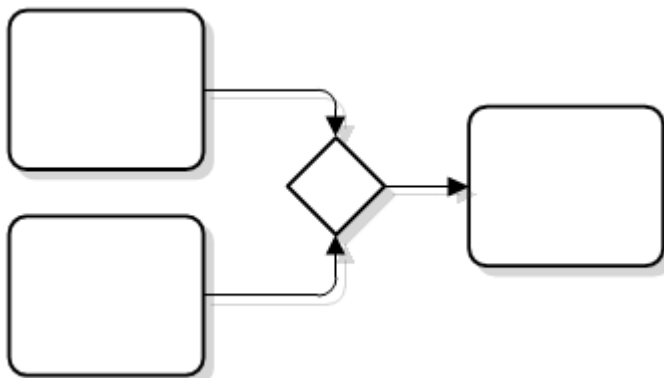


Figure 7.: BPMN Merge

#### 7.6.9.3.7. Sequence Flow Looping

Loops can be created by connecting a Sequence Flow to an “upstream” object. An object is considered to be upstream if that object has an outgoing Sequence Flow that leads to a series of other Sequence Flow, the last of which is an incoming Sequence Flow for the original object.

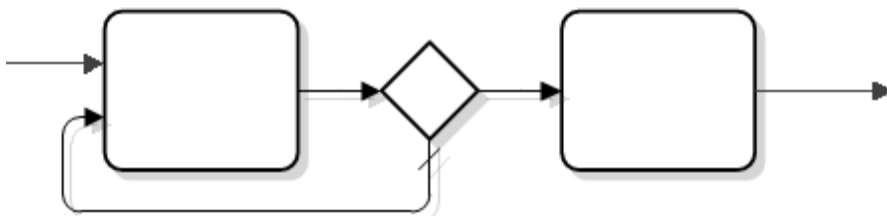


Figure 7.: Sequence Flow Looping

#### 7.6.9.3.8. Activity Looping



The attributes of Tasks and Sub- Processes will determine if they are repeated or performed once. There are two types of loops: Standard and Multi-Instance (see section 7.6.13). For Standard a small looping indicator will be displayed at the bottom-center of the activity. For Multi-Instance a small parallel indicator will be displayed at the bottom-center of the activity.



Figure 7.: Activity Looping

#### 7.6.9.3.9. Off-Page Connector

Generally used for printing, this object will show where the Sequence Flow leaves one page and then restarts on the next page. A Link Intermediate Event can be used as an Off-Page Connector.



Figure 7.: Off-Page Connector

#### 7.6.9.3.10. Exclusive Event-Based Gateway to Start a Process

**Event Gateways** can be used to instantiate a **Process**. By default the **Gateway's** `instantiate` attribute is *false*, but if set to *true*, then the **Process** is instantiated when the first **Event** of the **Gateway's** configuration is triggered.

- If the **Event Gateway's** `instantiate` attribute is set to *true*, then the marker for the **Event Gateway** looks like a **Multiple Start Event**.



In order for an **Event Gateway** to instantiate a **Process**, it must meet one of the following conditions:

- The **Process** does not have a **Start Event** and the **Gateway** has no *incoming Sequence Flow*, or
- The *incoming Sequence Flow* for the **Gateway** has a source of a **None Start Event**.
  - Note that no other *incoming Sequence Flow* are allowed for the **Gateway** (in particular, a loop connection from a downstream object).

#### 7.6.9.3.11. Parallel Event-Based Gateway to start a Process



In some situations a modeler may want the **Process** to be instantiated by one of a set of **Messages** while still requiring all of the **Messages** for the working of the same **Process** instance. To handle this, there is another variation of the **Event Gateway**.

- If the **Event Gateway's** `instantiate` attribute is set to *true* and the `eventGatewayType` attribute is set to *Parallel*, then the marker for the **Event Gateway** looks like a **Parallel Multiple Start Event**.
  - The **Event Gateway's** `instantiate` attribute **MUST** be set to *true* in order for the `eventGatewayType` attribute to be set to *Parallel* (i.e., for **Event Gateway's** that do not instantiate the **Process** **MUST** be *Exclusive*—a standard **Parallel Gateway** can be used to include parallel **Events** in the middle of a **Process**).

The **Parallel Event Gateway** is also a type of race condition. In this case, however, when the first **Event** is triggered and the **Process** is instantiated, the other **Events** of the **Gateway** configuration are not disabled. The other **Events**

are still waiting and are expected to be triggered before the **Process** can (normally) complete. In this case, the **Messages** that trigger the **Events** of the **Gateway** configuration must share the same correlation information.

### 7.6.10. InputSets

The InputSets attribute defines the data requirements for input to the activity. Zero or more InputSets MAY be defined. Each InputSet is sufficient to allow the activity to be performed (if it has first been instantiated by the appropriate signal arriving from an incoming Sequence Flow).

In BPMN 1.1 the element Input was replaced by ArtifactInputs and PropertyInputs.

For BPMN2.0 ArtifactInput is deprecated and the details of the Input element used to reference a DataObject.

Zero or more ArtifactInputs MAY be defined for each InputSet. For the combination of ArtifactInputs and PropertyInputs, there MUST be at least one item defined for the InputSet. An ArtifactInput is an Artifact, usually a DataObject. Note that the Artifacts MAY also be displayed on the diagram and MAY be connected to the activity through an Association--however, it is not required for them to be displayed.

Zero or one DataObject in the same scope MAY be referenced by its Id. One of DataObjectId OR ArtifactId is required.

Zero or more PropertyInputs MAY be defined for each InputSet. For the combination of ArtifactInputs and PropertyInputs, there MUST be at least one item defined for the InputSet.

Schema in XSD file.

	Description
ArtifactId	See section 6.4.7. and 7.1.9. Deprecated in BPMN2.0
ArtifactInputs	A list of Artifacts. Deprecated in BPMN2.0
DataObjectId	Reference the Id of a DataObject defined in the same scope.
PropertyInputs	A list of Properties.
RequiredForStart	The default value for this attribute is True. This means that the Input is required for an activity to start. If set to False, then the activity MAY start with the input if it is available, but MAY accept the input (more than once) after the activity has started. An InputSet may have some ArtifactInputs that have this attribute set to True and some that are set to False.

*Table : Input*

### 7.6.11. OutputSets

The OutputSets attribute defines the data requirements for output from the activity. Zero or more OutputSets MAY be defined. At the completion of the activity, only one of the OutputSets may be produced. It is up to the implementation of the activity to determine which set will be produced. However, the IORules attribute MAY indicate a relationship between an OutputSet and an InputSet that started the activity.

One or more Outputs MUST be defined for each OutputSet. An Output is an Artifact, usually a Document Object. Note that the Artifacts MAY also be displayed on the diagram and MAY be connected to the activity through an Association. However, it is not required for them to be displayed.

Schema in XSD file.

	Description
ArtifactId	See section 6.4.7. and 7.1.9. Deprecated in BPMN2.0.
Output	DataObjectId.

*Table : Output*

### 7.6.12. Transaction

A Sub-Process activity, whether it is Reusable (implemented by subflow) or embedded (block activity), can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol (such as BTP or WSTransaction).

There are three basic outcomes of a Transaction:

- Successful completion: this will be shown as a normal Sequence Flow that leaves the Sub-Process.
- Failed completion (Cancel):

When a Transaction is cancelled, then the activities inside the Transaction will be subjected to the cancellation actions, which could include rolling back the process and compensation for specific activities. Note that other mechanisms for interrupting a Sub-Process will not cause Compensation (e.g., Error, Timer, and anything for a non-Transaction activity). A Cancel Intermediate Event, attached to the boundary of the activity, will direct the flow after the Transaction has been rolled back and all compensation has been completed. The Cancel Intermediate Event can only be used when attached to the boundary of a Transaction activity. It cannot be used in any Normal Flow and cannot be attached to a non-Transaction activity. There are two mechanisms that can signal the cancellation of a Transaction:

- A Cancel End Event is reached within the Transaction Sub-Process. A Cancel End Event can only be used within a Sub-Process that is set to a Transaction.
- A Cancel Message can be received via the Transaction Protocol that is supporting the execution of the Sub-Process.
- Hazard: This means that something went terribly wrong and that a normal success or cancel is not possible. We are using an Error to show Hazards. When a Hazard happens, the activity is interrupted (without Compensation) and the flow will continue from the Error Intermediate Event.

The behavior at the end of a successful Transaction Sub-Process is slightly different than that of a normal Sub-Process. When each path of the Transaction Sub-Process reaches a non-Cancel End Event(s), the flow does not immediately move back up to the higher-level Parent Process, as does a normal Sub-Process. First, the transaction protocol must verify that all the participants have successfully completed their end of the Transaction. Most of the time this will be true and the flow will then move up to the higher-level Process. But it is possible that one of the participants may end up with a problem that causes a Cancel or a Hazard. In this case, the flow will then move to the appropriate Intermediate Event, even though it had apparently finished successfully.

Note: The exact behavior and notation for defining Transactions is still an open issue.

Schema in XSD file.

	Description
TransactionId	The TransactionId attribute provides an identifier for the Transactions used within a package/diagram.
TransactionMethod	TransactionMethod is an attribute that defines the technique that will be used to undo a Transaction that has been cancelled. The default is Compensate, but the attribute MAY be set to Store or Image.
TransactionProtocol	This identifies the Protocol (e.g., WS-Transaction or BTP) that will be used to control the transactional behavior of the Sub-Process.

Table : Transaction

### 7.6.13. Loop

The attributes of Tasks and Sub-Processes will determine if they are repeated or performed once. There are two types of loops: Standard and Multi-Instance.

A Standard Loop activity will have a boolean expression that is evaluated after each cycle of the loop. If the expression is still True, then the loop will continue. There are two variations of the loop, which reflect the programming constructs of while and until. That is, a while loop will evaluate the expression before the activity is performed, which means that the activity may not actually be performed. The until loop will evaluate the expression after the activity has been performed, which means that the activity will be performed at least once.

Multi-Instance loops reflect the programming construct foreach. The loop expression for a Multi-Instance loop is a numeric expression evaluated only once before the activity is performed. The result of the expression evaluation will be an integer that will specify the number of times that the activity will be repeated. There are also two variations of the Multi-Instance loop where the instances are either performed sequentially or in parallel.

Schema in XSD file.

	Description
LoopType	LoopType is an attribute and is by default None, but MAY be set to Standard or MultiInstance. If so, the Loop marker SHALL be placed at the bottom center of the activity shape. A Task of type Receive that has its Instantiate attribute set to True MUST NOT have a Standard or MultiInstance LoopType.
Standard	
LoopCondition	Attribute Deprecated(Use the LoopCondition Element). Standard Loops MUST have a boolean Expression to be evaluated, plus the timing when the expression SHALL be evaluated. LoopCondition can be either an attribute of type string or an element of type xpd:ExpressionType.
LoopCounter	This is updated at run time to count the number of executions of the loop and is available as a property to be used in expressions.
LoopMaximum	The Maximum an optional attribute that provides is a simple way to add a cap to the number of loops. This SHALL be added to the Expression defined in the LoopCondition.
TestTime	The expressions that are evaluated Before the activity begins are equivalent to a programming while function. The expression that are evaluated After the activity finishes are equivalent to a programming until function.
MultiInstance	
LoopCounter	The LoopCounter attribute is only applied for Sequential MultiInstance Loops and for processes that are being executed by a process engine. The attribute is updated at runtime by a process engine to count the number of loops as they occur. The LoopCounter attribute MUST be incremented at the start of a loop. Unlike a Standard loop, the modeler does not use this attribute in the MI_Condition Expression, but it can be used for tracking the status of a loop.
MI_Condition	Attribute deprecated (Use the MI_Condition element). MultiInstance Loops MUST have a numeric Expression to be evaluated--the Expression MUST resolve to an integer. MI_Condition can be either an attribute of type string or an element of type xpd:ExpressionType.
MI_Ordering	Sequential   Parallel This applies to only MultiInstance Loops. The MI_Ordering attribute defines whether the loop instances will be performed sequentially or in parallel. Sequential MI_Ordering is a more traditional loop. Parallel MI_Ordering is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use. If set to Parallel, the Parallel marker SHALL replace the Loop Marker at the bottom center of the activity shape.
MI_FlowCondition	None   One   All   Complex This attribute is equivalent to using a Gateway to control the flow past a set of parallel paths. An MI_FlowCondition of "None" is the same as uncontrolled flow (no Gateway) and means that all activity instances SHALL generate a token that will continue when that instance is completed. An MI_FlowCondition of "One" is the same as an Exclusive Gateway and means that the Token SHALL continue past the activity after only one of the activity instances has completed. The activity will continue its other instances, but additional Tokens MUST NOT be passed from the activity. An MI_FlowCondition of "All" is the same as a Parallel Gateway and means that the Token SHALL continue past the activity after all of the activity instances have completed. An MI_FlowCondition of "Complex" is similar to that of a Complex Gateway. The ComplexMI_FlowCondition attribute will determine the Token flow.

	Description
ComplexMI_FlowCondition Note that the attribute with this name is deprecated and replaced by the element with the same name.	If the MI_FlowCondition attribute is set to "Complex," then an Expression Must be entered. This Expression MAY reference Process data. The expression will be evaluated after each iteration of the Activity and SHALL resolve to a boolean. If the result of the expression evaluation is TRUE, then a Token will be sent down the activity's outgoing Sequence Flow. Otherwise, no Token for that iteration will be sent.

Table : Loop

## 7.7. Transition Information

The Transition Information describes possible transitions between activities and the conditions that enable or disable them (the transitions) during execution. Further control and structure restrictions may be expressed in the Activity definition.

Note that BPMN uses the term SequenceFlow for transition. See section 7.7.2.

A process definition is seen as a network of edges between the Activity nodes (i.e. as a process diagram). All edges are directed and given by a pair of Activities:

(From node, to node).

The edges of the Activity net may be labelled by **Transition conditions**. A Transition condition for a specific edge enables that transition if the condition evaluates to TRUE. If no routing condition is specified the Transition behaves as if a condition with value TRUE is present.

If there are multiple incoming or outgoing ("regular", see below) Transitions of an Activity, then further options to express control flow restrictions and condition evaluation semantics are provided in the Activity entity definition (AND/XOR variants of SPLIT/JOIN). Note: BPMN uses different terminology: Parallel, Inclusive and Exclusive.

A loop may be represented via a transition that returns to an Activity that was on a path that led to the transition. Typically, at least one of the activities in the loop will have multiple outgoing transitions, one or more of which will contain an exit condition from the loop.

For the identifiers and names defined in the Transition information the scope is the surrounding Process Definition.

It is possible to define or synchronize multiple (concurrent or alternative) control threads (split, join) and sequences of Transitions between Activities (cascading Transitions/conditions).

Schema in XSD file.

	Description
Assignments	See section 7.1.6.
Condition	A Transition condition expression based on relevant data field. (E.g. 'Contract' = 'SMALL' OR 'Contract' <\$20,000). Default: TRUE
ConnectorGraphicsInfos	See section 7.1.1.5.
Description	Short textual description of the Transition.
Extended Attributes	Optional extensions to meet individual implementation needs.
From	Determines the FROM source of a Transition. (Activity Identifier).
Id	Used to identify the Transition.
Name	Text used to identify the Transition.
Object	See section 7.1.9.4.
Quantity	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of Tokens that will be generated down the Sequence Flow.

	Description
To	Determines the TO target of a Transition (Activity Identifier).

*Table : Transition Information*

### 7.7.1. Condition

Schema in XSD file.

	Description
Type	<p>Define the type of transition condition, valid values are</p> <p><b>CONDITION</b>                      Indicates that the transition is to be executed if its condition is satisfied,</p> <p><b>OTHERWISE</b>                      Indicates that the transition is the default transition that is executed if no conditions are met.</p> <p><b>EXCEPTION</b>                      Indicates that the transition is to be executed if there is an exception and its condition is satisfied. NOT USED IN BPMN</p> <p><b>DEFAULTEXCEPTION</b>              Indicates that the transition is the default transition that is executed if there is an exception and no exception conditions are met. . NOT USED IN BPMN</p>
Expression	A condition expression represented via XML markup.

*Table : Condition*

#### 7.7.1.1. Exception Conditions

BPMN provides Intermediate Events (see 7.6.4.3) to handle exceptions. That is the preferred method.

The Exception and DEFAULTEXCEPTION types allow you to specify branches that are taken only when an Exception is raised. The EXCEPTION type is equivalent to the CONDITION type and the DEFAULTEXCEPTION matches the OTHERWISE type. The Condition may contain either the name of an Exception or a more complex expression. Except for the deadlines, exceptions are raised in an engine-specific manner. Like regular transitions, exception transitions are traversed only after the "From" activity has completed. An exception usually indicates abnormal completion. Note that BPMN provides Events to deal with Exceptions. See 7.6.4.3 Intermediate Events.

The following example illustrates a set of transitions from an activity that includes exceptions: branches 1 and 2 are processed under normal conditions; branches 3 and 4 are processed if there is an exception.

```

<Transitions>
  <Transition Id="branch1" From="CheckBalance" To="ProcessRequest">
    <Condition Type="CONDITION">Balance > 1000</Condition>
  </Transition>
  <Transition Id="branch2" From="CheckBalance" To="InsufFunds">
    <Condition Type="OTHERWISE"/>
  </Transition>
  <Transition Id="branch3" From="CheckBalance" To="MessageDisplay">
    <Condition Type="EXCEPTION">ATMDownException</Condition>
  </Transition>
  <Transition Id="branch4" From="CheckBalance" To="SendAlarm">
    <Condition Type="DEFAULTEXCEPTION"/>
  </Transition>
</Transitions>

```

### 7.7.2. BPMN view of Transition --- Sequence Flow

A Sequence Flow is used to show the order that activities will be performed in a Process. Each Flow has only one source and only one target. The source and target must be from the set of the following Flow Objects: Events (Start, Intermediate, and End), Activities (Task and Sub-Process), and Gateways. During performance (or simulation) of the

process, a Token will leave the source Flow Object, traverse down the Sequence Flow, and enter the target Flow Object.

#### 7.7.2.1. *Uncontrolled flow*



Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a gateway (7.6.2.1). The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flows that converge on or diverge from an activity. For each uncontrolled Sequence Flow a “Token” will flow from the source object to the target object.

BPMN does not use the term “Control Flow” when referring to the lines represented by Sequence Flow or Message Flow. The start of an activity is “controlled” not only by Sequence Flow (the order of activities), but also by Message Flow (a message arriving), as well as other process factors, such as scheduled resources. Artifacts can be Associated with activities to show some of these other factors. Thus, we are using a more specific term, “Sequence Flow,” since these lines mainly illustrate the sequence that activities will be performed.

#### 7.7.2.2. *Conditional flow*



A Sequence Flow MAY have a conditional expression attribute, depending on its source object. This means that the condition expression must be evaluated before a Token can be generated and then leave the source object to traverse the Flow. The conditions are usually associated with Decision Gateways, but can also be used with activities.

Condition expressions are evaluated at runtime to determine whether or not the flow will be used. If the conditional flow is outgoing from an activity, then the Sequence Flow will have a mini-diamond at the beginning of the line. If the conditional flow is outgoing from a Gateway, then the line will not have a mini-diamond (see Uncontrolled flow figure above).

#### 7.7.2.3. *Default flow*



For Data-Based Exclusive Decisions or Inclusive Decisions (section 7.6.2.2.1), one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flows are not true at runtime. This Sequence Flow will have a diagonal slash added to the beginning of the line.

By default, the ConditionType of a Sequence Flow is None. This means that there is no evaluation at runtime to determine whether or not the Sequence Flow will be used. Once a Token is ready to traverse the Sequence Flow (i.e., the Source is an activity that has completed), then the Token will do so. The normal, uncontrolled use of Sequence Flow, in a sequence of activities, will have a ‘None’ ConditionType. A ‘None’ ConditionType MUST NOT be used if the Source of the Sequence Flow is an Exclusive Data-Based or Inclusive Gateway. The ConditionType attribute MAY be set to Expression if the Source of the Sequence Flow is a Task, a Sub-Process, or a Gateway of type Exclusive-Data-Based or Inclusive. If the ConditionType attribute is set to Expression, then a condition marker SHALL be added to the line if the Sequence Flow is outgoing from an activity. However, a condition indicator MUST NOT be added to the line if the Sequence Flow is outgoing from a Gateway. An Expression ConditionType MUST NOT be used if the Source of the Sequence Flow is an Event-Based Exclusive Gateway, a Complex Gateway, a Parallel Gateway, a Start Event, or an Intermediate Event. In addition, an Expression ConditionType MUST NOT be used if the Sequence Flow is associated with the Default Gate of a Gateway. The ConditionType attribute MAY be set to Default only if the Source of the Sequence Flow is an activity or an Exclusive Data-Based Gateway. If the ConditionType is Default, then the Default marker SHALL be displayed..

Note that in the BPMN2.0 metamodel, default Gates are designated by attributes in the Activity or Gateway, not on the sequence flow itself as was done in BPMN1.x.

#### 7.7.2.4. *Exception flow*

Exception Flow occurs outside the Normal Flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process (7.6.4.3).

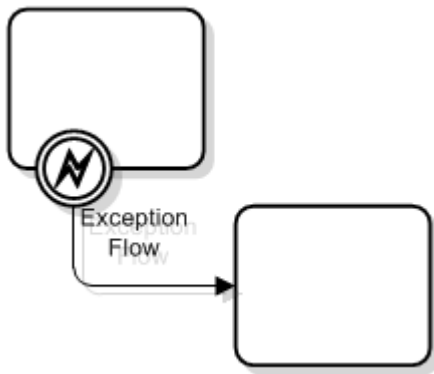


Figure 7.: Exception Flow

#### 7.7.2.5. Compensation Association

Compensation Association occurs outside the Normal Flow of the Process and is based upon an event (a Compensation Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event. The target of the Association must be marked as a Compensation Activity.

{Editorial comment: We are putting this under Sequence Flow because Associations have been described in BPMN as documentation not affecting the flow, whereas this seems otherwise}.

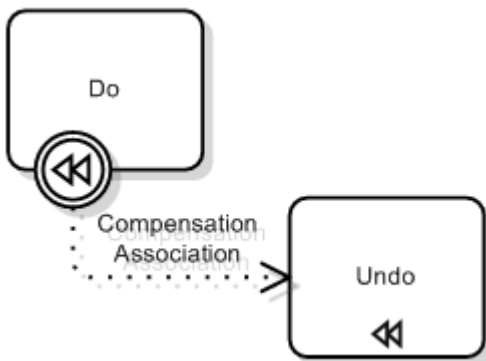


Figure 7.: Compensation Association

#### 7.7.2.6. Sequence Flow Rules

The Table below displays the BPMN Flow Objects and shows how these objects can connect to one another through Sequence Flow. The symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies and are not specified here. Refer to the sections for each individual object for more detailed information on the appropriate connection rules. Note that if a sub-process has been expanded within a Diagram, the objects within the sub-process cannot be connected to objects outside of the sub-process. Nor can Sequence Flow cross a Pool boundary.








































From\To						
						
						
						
						
						
						

Figure 7.: SequenceFlow Connection Rules

**Note** – Only those objects that can have incoming and/or outgoing Sequence Flow are shown in the table. Thus, Pool, Lane, Data Object, and Text Annotation are not listed in the table.

#### 7.7.2.7. SequenceFlow Examples

##### 7.7.2.7.1. Controlling Flow Across Processes

There may be situations within a Process where the flow is affected by or dependent on an activity that occurs in another Process. These events or conditions can be referred to as milestones. The process model must be able to identify and react to the milestone. That is, the continuation of a Process may be triggered by Signal Events, (see Figure below). In this example the activity at D cannot begin until the activity at B is completed. The upper subprocess THROWS the trigger and the lower subprocess CATCHES it.

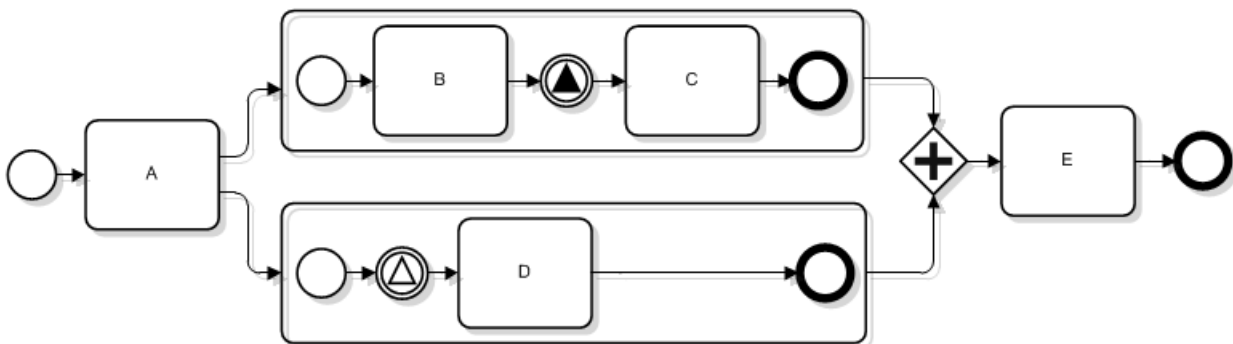


Figure 7. Signal Events Used to Synchronize Behavior Across Processes

## 7.8. Partner Links

Partner links are used to define a communication link between two parties, each assuming a role in the communication. From a modeling perspective the roles are normally, but not always, given by the name of a pool or lane. If used this way, the message flow between two pools will correspond to a partner link with each role corresponding to the name of the corresponding pool.

Partner links are optional and normally used to model the communication at an abstract level using WSDL and port types. A `WebServiceOperation` (see 7.9.6) can use a partner link for abstract modeling or service when a concrete web service is used and so port name is used instead of port type.

Partner links are defined in two levels. At the Package level, the partner link type defines a partner link name and one or two roles. The basic information about the partner link is defined at this level. At the process level, the partner link itself is defined using the partner link type. This allows partner link types to be reusable at the package level.

### 7.8.1. Partner Link Type

Partner link type defines the general information about a partner link at package scope. This allows multiple processes to use the same partner link type.

Schema in XSD file.

	Description
Role	A partner link may have one or two roles. In most cases they will correspond to the name of the two pools (or lanes) that are interchanging messages.
RoleName	The name of the role.
PortType	The port type implemented by the role. Corresponds to the WSDL port type.
Name	Name of the partner link type.
Id	Id of the partner link.

Table : *PartnerLinkType*

### 7.8.2. Partner Link

Partner link is used in a process and refers to a partner link type. It defines the role the process will play and the role the partner will use. `MyRole` element defines the role the process is playing and the `PartnerRole` element defines the role the partner is playing.

Schema in XSD file.

	Description
MyRole	Defines the role the process is playing in the interaction with the partner.
MyRole RoleName	Must match one of the two roles defined in the partner link type.
PartnerRole	Defines the role the partner is playing in the interaction with this process.
EndPoint	The end point for the partner. It may be the WSDL end point or the partner service listener end point.
PartnerRole RoleName	Must match one of the two roles defined in the partner link type.
ServiceName	The service name implemented by the partner and defined in the WSDL.
PortName	The port name implemented by the partner and defined in the WSDL.
Name	Name of this partner link. It may correspond to the name of the partner link type, but it is not required to be the same.
Id	Id of the partner link.

*Table : PartnerLink*

## 7.9. Messaging

Messages in XPDL are based on the WSDL model, and so, are not restricted to web services. When modeling and defining messaging they can be defined abstract in which case partner links should be used, or concrete in which case service should be used (see 7.9.6).

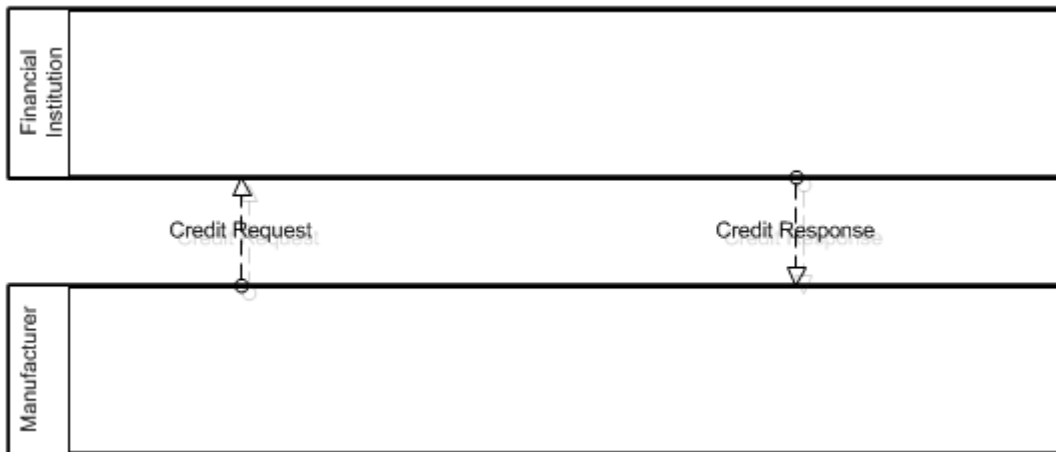
BPMN2.0 introduces explicit representation of the Message on a Message Flow. Not discussed in this document.

### 7.9.1. Message Flow

A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two entities. Thus, Message Flow **MUST** connect two Pools, either to the Pools themselves or to Flow Objects within the Pools. They cannot connect two objects within the same Pool.

### 7.9.2. BPMN Graphics and Semantics for Message Flow

The Message Flow can connect directly to the boundary of a Pool (See Figure below), especially if the Pool does not have any process details within (e.g. is a “Black Box”).



*Figure 7.: Message Flow connecting to the boundaries of two Pools*

A Message Flow can also cross the boundary of a Pool and connect to a Flow Object within that Pool (see Figure below).

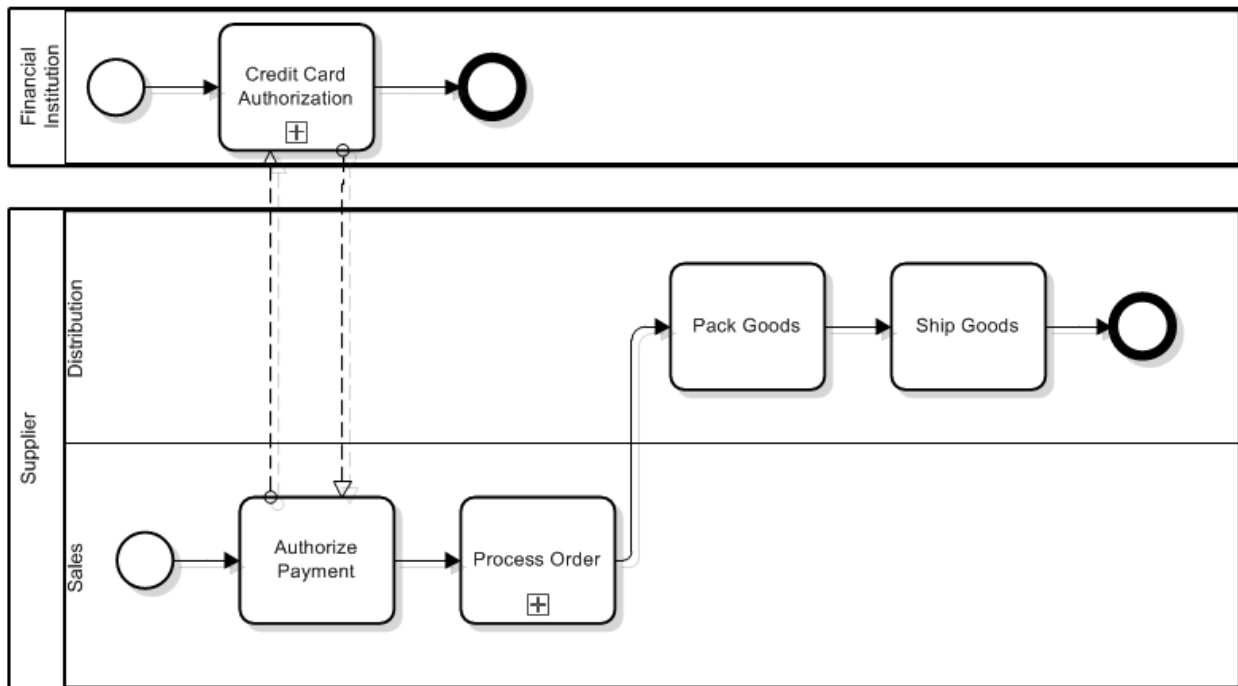


Figure 7.: Message Flow connecting to Flow Objects within two Pools

If there is an Expanded Sub-Process in one of the Pools, then the message flow can be connected to either the boundary of the Sub-Process or to objects within the Sub-Process.

### 7.9.3. Schema for Message Flow.

Schema in XSD file.

	Description
ConnectorGraphicsInfo	See section 7.1.1.5.
Description	Short textual description of the Message.
ExtendedAttributes	Optional extensions to meet individual implementation needs
Source	Determines the source of a MessageFlow (Activity or Pool).
Id	Used to identify the MessageFlow.
Message	Message is an optional attribute that identifies the Message that is being sent. See section 7.9.4.
Editors note: in BPMN1.1 MessageRef	
Name	Text used to identify the MessageFlow.
Object	See section 7.1.9.4.
Target	Determines the target of a MessageFlow (Activity or Pool).

Table : MessageFlow

### 7.9.4. Message Type

The Message type element is used in the definition of attributes for a Start Event, End Event, Intermediate Event, Task, Message Flow, etc.

Schema in XSD file.

	Description
Id	Id of the message.
Name	Text description of the Message.
From	Optional, but if present must be the name of a Participant (see section 7.4.1) /Process.
To	Optional, but if present must be the name of a Participant (see section 7.4.1) /Process.
Actual Parameters	Deprecated in XPDL2.2, use the Actual OParameters at the Activitiy Level. A list of parameters that compose the message. See section 7.1.6.1.
DataMappings	Deprecated in XPDL2.2, use the Actual OParameters at the Activitiy Level. Alternative approach to build the message. See section 7.6.5.4.7.
FaultName	When the message is an error message (for example an error response to a request), the FaultName corresponds to the fault (exception). See WebServiceFaultCatch to handle the error in the receiving end.

*Table : Message – attributes*

### 7.9.5. End Point

The end point can be a service (the URL of the listener implementing the service), or a WSDL (the URL of the WSDL location).

Schema in XSD file.

	Description
ExternalReference	The URL of the end point.
EndPointType	Type of end point: <div> <div>WSDL</div> <div>When the end point corresponds to the location of a WSDL file.</div> </div> <div> <div>Service</div> <div>When the end point corresponds to the address of a listener implementing the service.</div> </div>

*Table : End Point*

### 7.9.6. Web ServiceOperation

A web services operation is defined by using a partner link or alternative describing the service.

Schema in XSD file.

	Description
Partner	Included only when a partner link is used to define this operation.
PartnerLinkId	Partner link to be used.
RoleType	The role in the partner link that is being used <div> <div>MyRole</div> <div>Using the process role</div> </div> <div> <div>PartnerRole</div> <div>Using the partner role</div> </div>
OperationName	The name of the operation implemented by the service as it is defined in the WSDL.
Service	Included only when a concrete service operation is being defined.
ServiceName	The name of the service implementing the operation.
PortName	The port name in which the service is implementing the operation.
EndPoint	The end point implementing the service.

*Table : Web Service Operation***7.9.7. Web Service Fault Catch**





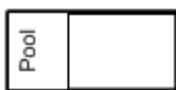












Used to catch web services faults and to either execute a block activity or a transition.

Schema in XSD file.

	Description
Message	Optional message that may contain information about the fault.
BlockActivity	Block activity that will be executed if the WebServiceFaultCatch is activated by a fault.
TransitionRef	TransitionRef that will be executed if the WebServiceFaultCatch is activated by a fault.
FaultName	Name of the fault that will be caught by this WebServiceFaultCatch. If the name is not provided, then the WebServiceFaultCatch will catch any fault generated by the web service.

*Table : Web Service Fault Catch***7.9.8. Message Flow Rules**

The Table below displays the BPMN modeling objects and shows how these objects can connect to one another through Message Flow. The symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies that are not specified here. Refer to the sections for each individual object for more detailed information on the appropriate connection rules. Note that Message Flow cannot connect to objects that are within the same Pool.

From \ To						
						
						
						
						
						
						

*Figure 7.: Message Flow Connection Rules*

**Note** – Only those objects that can have incoming and/or outgoing Message Flow are shown in the table. Thus, Lane, Gateway, Data Object, and Text Annotation are not listed in the table.

## 7.10. Association

An Association is used to associate information and Artifacts with Flow Objects. Text and graphical non-Flow Objects can be associated with the Flow Objects and Flow. An Association is also used to show the activities used to compensate for an activity.

An Association is also used to associate Data Objects with other objects. A Data Object is used to show how documents are used throughout a Process. Refer to section 7.1.9.5 for more information on Data Objects. Note that in BPMN2.0 Data Objects are changed from Artifacts to Flow Objects. We treat the BPMN2.0 Data Objects in a separate section.

### 7.10.1. BPMN Graphics and Semantics

.....

#### An Association

If there is a reason to put directionality on the association then:

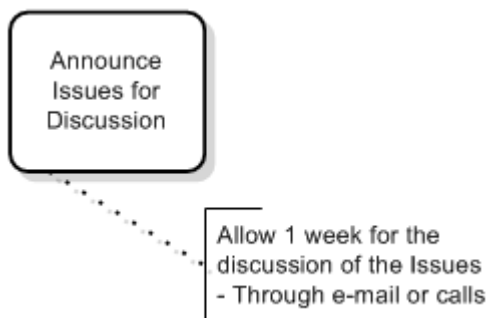
- A line arrowhead MAY be added to the Association line. (see Figure below).

A directional Association is often used with Data Objects to show that a Data Object is either an input to or an output from an activity.

.....>

#### A directional Association

An Association is used to connect user-defined text (an Annotation) with a Flow Object (see Figure below).



*Figure 7.: An Association of Text Annotation*

An Association is also used to associate BPMN 1.x Data Objects with other objects (see Figure below). A Data Object is used to show how documents are used throughout a Process.

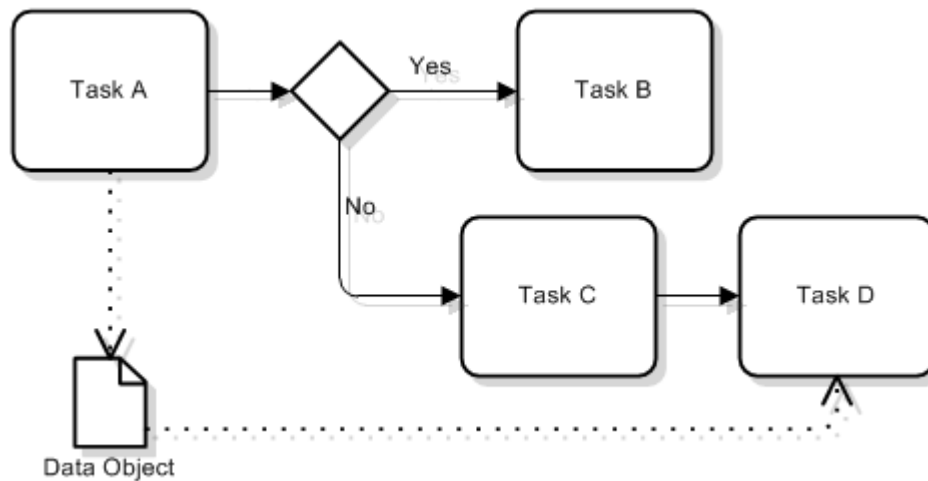


Figure 7.: An Association connecting a Data Object with a Flow

### 7.10.2. Schema for Association

Schema in XSD file.

	Description
Association Direction	None   To   From   Both
Connector Graphics Info	See section 7.1.1.5.
Description	Short textual description of the Association.
Source	Determines the source of an Association (any graphical object).
Id	Used to identify the Association.
Name	Text used to identify the Association.
Object	See section 7.1.9.4.
Target	Determines the target of an Association (any graphical object).

Table : Association

## 7.11. Participants

The Participant is one of the following types: resource set, resource, organizational unit, role, human, or system. A role and a resource are used in the sense of abstract actors. This definition is an abstraction level between the real performer and the activity, which has to be performed. During run time these abstract definitions are evaluated and assigned to concrete human(s) and/or program(s).

Note that this notion of Participant differs from the BPMN term (see section 7.4.1).

The scope of the identifier of a participant entity declaration in a minimal resource repository or organizational model is the surrounding entity (Process Definition or Process Model Definition) in which it is defined.

An external resource repository or organizational model may contain substantial additional information that complements the basic participant types presented in here.

Schema in XSD file.

	Description
Description	Short textual description of a participant.



	Description
ExternalReference	A reference to an external specification of a participant. See section 7.1.5.
ExtendedAttributes	Optional extensions to meet individual implementation needs.
Id	Used to identify the participant definition.
Name	Text used to identify a performer.
ParticipantType	Definition of the type of participant entity.

*Table : Participant*

### 7.11.1. Participant Entity Types

The Participant entity type attribute characterises the participant to be an individual, an organisational unit or an abstract resource such as a machine.

Schema in XSD file.

	Description
Type	<div>RESOURCE_SET</div> <div>A set of resources.</div> <div>RESOURCE</div> <div>A specific resource agent.</div> <div>ROLE</div> <div>This type allows performer addressing by a role or skill set. A role in this context is a function a human has within an organization. As a function isn't necessarily unique, a coordinator may be defined (for administrative purposes or in case of exception handling) and a list of humans the role is related to.</div> <div>ORGANIZATIONAL_UNIT</div> <div>A department or any other unit within an organizational model.</div> <div>HUMAN</div> <div>A human interacting with the system via an application presenting a user interface to the participant.</div> <div>SYSTEM</div> <div>An automatic agent.</div>

*Table : Participant Entity Type*

## 7.12. Relevant data field/Property

Relevant data fields represent the variables of a process or Package Definition. They are typically used to maintain decision data (used in conditions) or reference data values (parameters), which are passed between activities or subflow. This may be differentiated from application data, which is data managed or accessed wholly by the invoked applications and which is not accessible to the process or workflow management system. The relevant data field list defines all data objects which are required by the process. The attribute `DataType` explicitly specifies all information needed for a process or workflow management system to define an appropriate data object for storing data, which is to be handled by an active instance of the process.

Relevant data field can be defined in a process and in a Package. The scopes differ in that the former may only be accessed by entities defined inside that process, while the latter may be used also e.g. to define the parameters of a process entity.

Where parameters are passed to a called subflow outside the current model definition (e.g. to support remote process invocation) it is the responsibility of the process designer(s) to ensure that data type compatibility exists across the parameter set.

Schema in XSD file.

	Description
DataType	Data type of the process variable. See Section 7.13.
Description	Short textual description of the data defined.
ExtendedAttributes	Optional extensions to meet individual implementation needs.
Id	Used to identify the relevant data field.
InitialValue	Pre-assignment of data for run time.
IsArray	Indicates if it is an array.
Length	The length of the data.
Name	Text used to identify the relevant data field.
ReadOnly	The datafield or formal parameter is described as readOnly or as a constant and its value cannot be changed.
Correlation	Used in BPMN mapping to BPEL.

*Table : Relevant data field*

## 7.13. Data Types

Data types consist of a set of standard types that may be used as part of the data specification of relevant data field, formal parameters, and Processes. You can also declare a new data type within a TypeDeclaration and use it wherever the standard data types are used. A data type may be selected from the following set of types.

```
<xsd:element name="DataType">
  <xsd:complexType>
    <xsd:group ref="xpd:DataTypes"/>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:element>
```

Schema in XSD file.

	Description
ArrayType	A fixed size set of data all of the same data type (deprecated).
BasicType	A simple type: STRING, INTEGER, FLOAT, DATETIME, DATE, TIME, REFERENCE, BOOLEAN, or PERFORMER.
DeclaredType	A reference to a data type declared in a TypeDeclaration element.
EnumerationType	A set of legal values of a variable or parameter (deprecated).
ExternalReference	A reference to a type defined in an external document. See Section 7.1.5.
ListType	An unbounded set of data all of the same data type (deprecated).
RecordType	A set of members that may be of different types (deprecated).
SchemaType	A data type defined using an XML schema.
UnionType	A set of members only one of which will be used for an instance of the data (deprecated).

*Table : Standard Data Types*

### 7.13.1. Basic Data Types

Schema in XSD file.

	Description
STRING Type	A finite-length sequence of characters.
FLOAT Type	A floating point or double precision number. The maximum size of the number is not specified in XPDL.
INTEGER Type	A number represented by an optional sign followed by a finite-length sequence of decimal digits. The maximum size of the integer is not specified in XPDL.
REFERENCE Type	A reference to an external data type – now deprecated. The ExternalReference is the recommended way to refer to an external data type.
DATETIME Type	A specific instance of time. The date format is not specified within XPDL.
DATE Type	A specific date instance. This differs from DATETIME in that there is no time component.
TIME Type	A specific time instance. This differs from DATETIME in that there is no date component.
BOOLEAN Type	A data instance of a Boolean type is one having one of the values TRUE or FALSE. The internal representation of these values is not defined in XPDL.
PERFORMER Type	A data instance of a performer type is one having a value of a declared participant.

Table : Basic Data Types

### 7.13.2. Complex Data Types

XPDL permits the definition of complex data types such as arrays, records, unions, enumerations, and lists. Complex data types are defined using the SchemaType. The RecordType, UnionType, EnumerationType, ArrayType, and ListType, which were used in the past to define complex data, are now deprecated. They have been left in the XPDL schema for compatibility with previous versions.

#### 7.13.2.1. Schema Type

The SchemaType allows users to define a data type using XML schema syntax. It may also be used to define an XML string that should conform to the schema.

Schema in XSD file.

The following, for example, could describe a C++ or Java class, a C structure, or an XML string:

```
<SchemaType>
  <schema xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="PO">
      <complexType>
        <sequence>
          <element name="CustomerName" type="string"/>
          <element name="Address" type="string"/>
          <element name="OrderNumber" type="string"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</SchemaType>
```

#### 7.13.2.2. Record Type

Schema in XSD file.

	Description
Member	A field in the record.

	Description
DataTypes	Data type of a member. See Table : Standard Data Types.

*Table : Record Type***7.13.2.3. Union Type**

Schema in XSD.

	Description
Member	A field in the union.
DataTypes	Data type of a member. See Table : Standard Data Types.

*Table : Union Type***7.13.2.4. Enumeration Type**

Schema in XSD.

	Description
Enumeration Value	An element that represents one of the values in an enumeration.
Name	The name of the value.

*Table : Enumeration Type***7.13.2.5. Array Type**

Schema in XSD file.

	Description
DataTypes	The data type of array entries. See Table : Standard Data Types.
LowerIndex	The lower bound of an ArrayType.
UpperIndex	The upper bound of an ArrayType.

*Table : Array Type***7.13.2.6. List Type**

Schema in XSD file.

	Description
DataTypes	The data type of list entries. See Table : Standard Data Types.

*Table : List Type***7.13.3. Declared Data Types**

It is possible to reuse a complex data definition wherever you can use a standard XPDL type. Define the data type under a TypeDeclaration and then refer to it using the DeclaredType data type.

**7.13.3.1. Type Declaration**

Schema in XSD file.

	Description
DataTypes	The data type. See Table : Standard Data Types.
Description	An informal description of the data type.
ExtendedAttributes	Optional extensions to meet individual implementation needs.
Id	An identifier for the TypeDeclaration.
Name	The name of the TypeDeclaration.

*Table : Type Declaration*

Example to reuse a SchemaType to define a purchase order:

```

<TypeDeclarations>
  <TypeDeclaration Id="POType" Name="PurchaseOrder">
    <SchemaType>
      <schema xmlns="http://www.w3.org/2000/10/XMLSchema">
        <element name="PO">
          <complexType>
            <sequence>
              <element name="CustomerName" type="string"/>
              <element name="Address" type="string"/>
              <element name="OrderNumber" type="string"/>
            </sequence>
          </complexType>
        </element>
      </schema>
    </SchemaType>
  </TypeDeclaration>
</TypeDeclarations>

```

### 7.13.3.2.Declared Type

Schema in XSD file.

	Description
Id	A reference to a data type declared in a TypeDeclaration.
Name	A name for the declared type

*Table : Declared Data Type*

Example of using the DeclaredType type to define a number of POType variables.

```

<DataFields>
  <DataField Id="newPO">
    <DataType>
      <DeclaredType Id="POType"/>
    </DataType>
  </DataField>
  <DataField Id="checkedPO">
    <DataType>
      <DeclaredType Id="POType"/>
    </DataType>
  </DataField>
</DataFields>

```

## **8. XPDL Schema**

The full Schema for XPDL is contained in a separate document available here:

[http://www.xpdl.org/standards/xpdl-2.2/bpmnxd\\_40a.xsd](http://www.xpdl.org/standards/xpdl-2.2/bpmnxd_40a.xsd)

## 9. Figures and Tables

### 9.1. Figures

Figure 5.1: The Concept of the Process Definition Interchange .....	11
Figure 6.1: Package Definition Meta Model.....	14
Figure 6.2: Process Definition Meta Model.....	16
Figure 6.3: Swimlanes .....	17
Figure 6.4: Activities .....	19
Figure 6.5: BPMN Connections.....	19
Figure 6.6: BPMN Artifacts.....	20
Figure 7.1: Artifacts.....	30
Figure 7.2: A Data Object associated with a Sequence Flow .....	31
Figure 7.3: Data Objects shown as inputs and outputs .....	32
Figure 7.4: A Group Artifact .....	32
Figure 7.5: A Group around activities in different Pools.....	33
Figure 6: Data Object produced by one activity and consumer by another .....	35
Figure 7: Alternate visualization of Data Object associated directly with Sequence Flow .....	35
Figure 7.8: Simple Class.....	44
Figure 7.9: Pool .....	54
Figure 7.10: Message Flow connecting to the boundaries of two Pools.....	55
Figure 7.11: Message Flow connecting to Flow Objects within two Pools .....	55
Figure 7.12: Main (Internal) Pool without boundaries.....	56
Figure 7.13: Two Lanes in a Vertical Pool .....	57
Figure 7.14: Two Lanes in a Horizontal Pool.....	57
Figure 7.15: An Example of Nested Lanes .....	58
Figure 7.16: BPMN Activity Types.....	68
Figure 7.17: Activity Structures & Transition Conditions.....	69
Figure 7.18: Gateway Types.....	73
Figure 7.19: Exclusive Decision – Data Based.....	73
Figure 7.20: Exclusive Merge.....	74
Figure 7.21: Exclusive Decision without Indicator .....	76
Figure 7.22: Exclusive Decision with Indicator.....	76
Figure 7.23: Exclusive Merge Without the Indicator.....	77
Figure 7.24: Uncontrolled Sequence Flow .....	77
Figure 7.25: Exclusive Gateway is required to act as a merging object .....	78
Figure 7.26: An Event-Based Exclusive Gateway using Receive Tasks .....	79
Figure 7.27: An Event-Based Exclusive Gateway using Message Events.....	79

Figure 7.28: An Inclusive Decision using Conditional Sequence Flow .....	81
Figure 7.29: An Inclusive Decision using an Inclusive Gateway .....	82
Figure 7.30: An Inclusive Gateway Merging Sequence Flow .....	82
Figure 7.31: A Complex Decision (Gateway) .....	83
Figure 7.32: A Complex Merge (Gateway) .....	84
Figure 7.33: A Parallel Gateway.....	85
Figure 7.34: Joining – the joining of parallel paths .....	85
Figure 7.35: Three Types of Events.....	86
Figure 7.36: Intermediate Event on an Activity Boundary .....	92
Figure 7.37: Event Subtypes – Catching and Throwing .....	107
Figure 7.38: Task Markers.....	109
Figure 7.39: Collapsed and Expanded Subflow .....	115
Figure 7.40: Expanded Subprocess for parallel activities .....	116
Figure 7.41: SubProcess Markers .....	116
Figure 7.42: Reusable Subprocess Semantics: BPMN perspective .....	117
Figure 7.43: Subprocess behavior as a Transaction.....	118
Figure 7.44: BPMN Fork.....	126
Figure 7.45: BPMN Join.....	127
Figure 7.46: BPMN Exclusive Data-Based Decision .....	127
Figure 7.47: BPMN Event-Based Decision .....	127
Figure 7.48: BPMN Inclusive Decision.....	128
Figure 7.49: BPMN Merge .....	128
Figure 7.50: Sequence Flow Looping.....	128
Figure 7.51: Activity Looping .....	129
Figure 7.52: Off-Page Connector.....	129
Figure 7.53: Exception Flow .....	136
Figure 7.54: Compensation Association .....	136
Figure 7.55: SequenceFlow Connection Rules .....	137
Figure 7.56 <b>Signal Events Used to Synchronize Behavior Across Processes</b> .....	137
Figure 7.57: Message Flow connecting to the boundaries of two Pools.....	139
Figure 7.58: Message Flow connecting to Flow Objects within two Pools .....	140
Figure 7.59: Message Flow Connection Rules .....	142
Figure 7.60: An Association of Text Annotation.....	143
Figure 7.61: An Association connecting a Data Object with a Flow .....	144



## 9.2. Tables

Table 1: Pages.....	23
Table 2: Node Graphics Info .....	24
Table 3: Connector Graphics Info .....	25
Table 4: ExpressionType .....	25
Table 5: Extended Attributes .....	26
Table 6: Formal Parameters .....	27
Table 7: External Reference .....	27
Table 8: Assignment .....	29
Table 9: Category .....	29
Table 10: Artifact.....	30
Table 11: Object .....	31
Table 12: Data Object.....	32
Table 13: Group.....	33
Table 14: Package Definition .....	40
Table 15: Package Definition Header – Attributes .....	41
Table 16: Vendor Extension -- Attributes.....	41
Table 17: Redefinable Header .....	42
Table 18: Conformance Class Declaration .....	45
Table 19: Script .....	50
Table 20: External Package Reference .....	51
Table 21: Application Declaration.....	52
Table 22: EJB Application Type .....	52
Table 23: POJO Application Type.....	53
Table 24: XSLT Application Type .....	53
Table 25: Script Application Type.....	53
Table 26: WebService Application Type.....	53
Table 27: BusinessRule Application Type .....	53
Table 28: Form Application Type .....	54
Table 29: Pools .....	56
Table 30: Lane .....	58
Table 31: Process Definition.....	61
Table 32: Process Definition Header .....	62
Table 33: Process Redefinable Header .....	63
Table 34: ActivitySet.....	64
Table 35: Entity type relationships for different Activity types.....	68
Table 36: Process Activity .....	71
Table 37: Route Activity .....	72
Table 38 BlockActivity.....	86

Table 39: Event.....	87
Table 40: Start Event subtypes .....	90
Table 41: Start Event Activity .....	92
Table 42: Intermediate Event sub types.....	96
Table 43: Intermediate Event Activity.....	98
Table 44: End Event subtypes .....	100
Table 45: End Event Activity .....	101
Table 46: Event Trigger Result Compensation.....	102
Table 47: Event Result Error .....	102
Table 48: Event Result Multiple .....	103
Table 49: Event Trigger Result Link .....	103
Table 50: Event Trigger Result Message.....	103
Table 51: Trigger Result Signal.....	104
Table 52: Event Trigger Intermediate Multiple .....	104
Table 53: Event Trigger Multiple .....	104
Table 54: Trigger Conditional .....	105
Table 55: Event Trigger Timer .....	105
Table 56: Implementation Alternatives .....	108
Table 57: Task Manual .....	109
Table 58: Task Receive .....	110
Table 59: TaskReference .....	110
Table 60: TaskSend .....	111
Table 61: Task Service .....	111
Table 62: Task Script.....	112
Table 63: Task User.....	112
Table 64: Tool .....	113
Table 65: SubFlow.....	115
Table 66: DataMapping .....	120
Table 67: Reference .....	121
Table 68: Deadline.....	123
Table 69: Simulation Information.....	124
Table 70: Transition Restrictions.....	124
Table 71: Join .....	125
Table 72: Split .....	126
Table 73: Input.....	130
Table 74: Output.....	130
Table 75: Transaction .....	131
Table 76: Loop.....	133
Table 77: Transition Information.....	134

Table 78: Condition .....	134
Table 79: PartnerLinkType .....	138
Table 80: PartnerLink .....	139
Table 81: MessageFlow .....	140
Table 82: Message – attributes .....	141
Table 83: End Point .....	141
Table 84: Web Service Operation .....	142
Table 85: Web Service Fault Catch .....	142
Table 86: Association .....	144
Table 87: Participant .....	145
Table 88: Participant Entity Type .....	145
Table 89: Relevant data field .....	146
Table 90: Standard Data Types .....	146
Table 91: Basic Data Types .....	147
Table 92: Record Type .....	148
Table 93: Union Type .....	148
Table 94: Enumeration Type .....	148
Table 95: Array Type .....	148
Table 96: List Type .....	148
Table 97: Type Declaration .....	149
Table 98: Declared Data Type .....	149