4. Business Process Diagram Graphical Objects

This section details the graphical representation and the semantics of the behavior of BPD elements.

4.1 Common BPD Object Attributes

The following table displays a set of common attributes for BPMN objects (specifically Events, Activities, and Gateways):

Attributes	Description
Id: String	This is a unique ID that identifies the object from other objects within the diagram.
Name: String	Name is an attribute that is text description of the object.
Assign *: Expression	Zero or more assignments MAY be made for the object. The expressions SHALL be evaluated when the flow of the Process (the Token) arrives at the object.
Pool ?: PoolName	If Pools are used, then the PoolName MUST be added to the object to identify its location.
Lane *: LaneName	If the Pool has more than one Lane, then a LaneName MUST be added. There MAY be multiple Lanes listed if the Lanes are organized in matrix or overlap in a non-nested manner.
Documentation ?: String	The modeler MAY add optional text documentation about the object.

Table 5 Common Object Attributes

4.2 Events

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 or an impact. 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 Intermediate Events have “Triggers” that define the cause for the event. There are multiple ways that these events can be triggered (refer to the section entitled “Start Event Triggers” on page 36 and “Intermediate Event Triggers” on page 50). 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 (refer to the section entitled “End Event Results” on page 42).

All Events share the same shape footprint, a small circle. Different line styles, as shown below, 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.

4.2.1 Start

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

The Start Event shares the same basic shape of the Intermediate Event and End Event, a circle, but is drawn with a single thin line (see Figure 1). Text associated with the Start Event (e.g., its name) can be placed above or below the shape, in any direction or location, or on the outgoing Sequence Flow, depending on the preference of the modeler or modeling tool vendor.

Figure 1 A Start Event

Throughout this document, we will discuss how Sequence Flow proceeds within a Process. To ~~facillitate~~ facilitate this discussion, we will employ the concept of a "**Token**" that will traverse the Sequence Flows and pass through the flow objects in the Process. The behavior of the Process can be described by tracking the path(s) of the Token through the Process. A Token will have a unique identity, called a TokenID set, that can be used to distinguish multiple Tokens that may exist because of concurrent Process instances or the dividing of the Token for parallel processing within a single Process instance. The parallel dividing of a Token creates a lower level of the TokenID set. The set of all levels of TokenID will identify a Token. The TokenID set for a Token will be in the following format: "TokenID.TokenID. … TokenID," each level being separated by a dot.

A Start Event generates a Token that must eventually be consumed at an End Event (which may be implicit if not graphically displayed). The path of Tokens MUST be explicitly traced through the network of Sequence Flow with a Process. There CANNOT be any implicit flow during the course of normal Sequence Flow. Tokens can also be consumed through exception handling Intermediate Events, which act like a forced end to a Process level. Note: A Token does not traverse the Message Flows since it is a Message that is passed down those Flows (as the name implies).

Semantics of the Start Event include:

- ~~*This shape*~~ *A Start Event is optional*~~—~~: a Process ~~level (~~level—a top-level Process or an expanded Sub-Process~~)~~ —MAY (is not required ~~to~~ to) have ~~this shape~~a Start Event:
  - If a Process is complex and/or the starting conditions are not obvious, then a Start Event SHOULD be used.
  - If there is ~~a Start~~ an End Event, then there ~~has to~~ MUST be at least one ~~End~~ Start Event.

- If the Start Event is used, then there ~~can~~ SHALL NOT be ~~no~~ other flow elements that do not have incoming Sequence Flow (of those elements that can accept Sequence Flow)—all other flow objects ~~must~~ MUST be a target of at least one Sequence Flow.
  - An exception to this is the Intermediate Event, which ~~can~~ MAY be without an incoming Sequence ~~Flow~~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) ~~will~~ 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.

- There ~~can~~ MAY be multiple Start Events for a given Process level~~. Each Start Event is an independent event~~.
  - Each Start Event is an independent event. That is, a Process Instance SHALL be generated when the Start Event is triggered.

Note: A BPD may have more than one Process level (i.e., it can include Expanded Sub-Processes). The use of Start and End Events is independent for each level of the diagram.

That is, when the trigger for a Start Event occurs, Tokens will be generated for each outgoing Sequence Flow from that event. The TokenID set for each of the Tokens will be established such that it can be identified that the Tokens are all from the same parallel Fork (AND-Split) and the number of Tokens in the group. 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 (refer to the section entitled “Attributes” on page 61 (for sub-processes) and “Attributes” on page 69 (for Tasks) for more information about activity attributes). In addition, a correlation mechanism will be required so that different triggered Start Events will apply to the same process instance. Correlation will likely be handled through Event attributes, but this an open issue will be addressed in a later version of the specification. Refer to the section entitled “Open Issues” on page 165 for a complete list of the issues open for BPMN.

- ~~A sub-process, which is a process within a process, if expanded to show its detail, can also have Start Events.~~

Start Event Triggers

There are many ways that can 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. There are six types of Start Events in BPMN: None, Message, Timer, Rule, Link, and Multiple.

Table 6 displays the types of Triggers and the graphical marker that will be used for each:

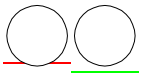
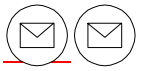
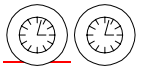
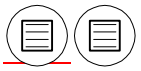
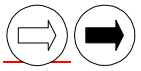
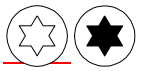
Trigger	Description	Marker
None	The modeler does not display the type of Event. It is also used for a Sub-Process that starts when the flow is triggered by its Parent Process.	
Message	A message arrives from a participant and triggers the start of the Process.	
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process.	
Rule	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.	
Link	A Link is a mechanism for connecting the end (Result) of one Process to the start (Trigger) of another. Typically, these are two Sub-Processes within the same parent Process.	
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.	

Table 6 Start Event Types

Attributes

The following are ~~identified~~ attributes of a Start Event, which extends the set of common object elements (see Table 5):

Attribute	Description
Name ?	Name is an optional property that is text description of the Event.
Trigger (None | Message | Timer | Rule | Link | Multiple) : None	Trigger is a property (default None) that defines the type of trigger expected for that Start.
Message: MessageName	If the Trigger is a Message, then the name of the Message must be supplied.
Timer: (Timedate | TimeCycle): Timedate	If the Trigger is a Timer, then a timedate or a timedatecycle must be entered.
Rule: RuleExpression	If the Trigger is a Rule, then an expression must be entered.
Link: LinkName	If the Trigger is a Link, then the name of the Link must be supplied.
Multiple: Trigger + (except Multiple)	If the Trigger is a Multiple, then a list of the Trigger must have the appropriate data.
Assign *: Expression	Zero or more assignments can be made. Each assignment is an expression.
OutgoingSequenceFlow +: SequenceFlowName	One or more outgoing Sequence Flows can be idenitified for the Start Event.

Attribute	Description
IncomingMessageFlow *: MessageFlowName	Zero or more incoming Message Flows can be idenitified for the Start Event, but the Trigger type must be Message or Multiple (with Message as one of Trigger types). For a Message Flow there will be an associated BPEL4WS *receive* or a BPML *one-way action* to receive the message. For multiple Message Flows, there will be BPEL4WS *pick* or BPML *choice* that will receive only one of the messages.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Start Event to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the Start Event.
Documentation ?	The modeler can add optional text documentation about the Start Event.

Attribute	Description
Trigger (None | Message | Timer | Rule | Link | Multiple) : None	Trigger is an attribute (default None) that defines the type of trigger expected for that Start.
Message: MessageName	If the Trigger is a Message, then the name of the Message MUST be supplied.
Timer: (Timedate | TimeCycle): Timedate	If the Trigger is a Timer, then a timedate or a timedatecycle MUST be entered.
Rule: RuleExpression	If the Trigger is a Rule, then an expression MUST be entered.
Link: LinkName	If the Trigger is a Link, then the name of the Link MUST be supplied.
Multiple: Trigger + (except Multiple)	If the Trigger is a Multiple, then a list of the Triggers MUST have the appropriate data (as defined above).

Table 7 Start Event Attributes

Sequence Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how they may be source or targets of Sequence Flows.

- A Start Event SHALL NOT be a target for a Sequence Flow; it MUST NOT have incoming Sequence Flows.
- A Start Event ~~cannot~~ MUST be a ~~target~~ source for a Sequence Flow~~; there can be no incoming Sequence Flows~~.
- ~~A Start Event can be a source for a~~ Multiple Sequence ~~Flow; multiple Sequence~~ Flows ~~can~~ MAY originate from a Start Event. For each Sequence Flow that has the Start Event as a source, ~~there will be~~ a new parallel path SHALL be generated~~.~~.
  - When a Start Event is not used, then all flow objects that do not have an incoming Sequence Flow ~~will~~ SHALL be the start of a separate parallel path~~.~~.

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

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how they may be source or targets of Sequence Flows.

Note: All Message Flows described here 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 ~~can~~ 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~~. to see how the incoming Message Flows are mapped to BPEL4WS and BPML elements~~.
  - The trigger ~~property~~ attribute of the Start Event ~~must~~ MUST be set to ~~Message~~ “Message” or ~~Multiple~~ “Multiple” if there are any incoming Message Flows.
- A Start Event ~~cannot~~ SHALL NOT be a source for a Message Flow; it ~~can~~ MUST NOT have ~~no~~ outgoing Message Flows.

Mapping to Execution Languages

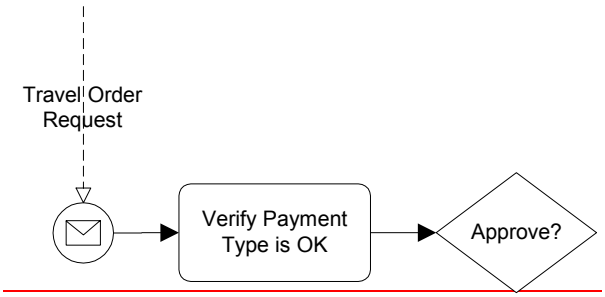
~~The following two sections describe how the use of Start Events will map to BPEL4WS and BPML, respectively.~~

~~**BPEL4WS**~~

- ~~If the Start Event has an expression for the assign property, then this will map to a BPEL4WS *assign*.~~

~~Each type of Start Event Trigger will have a different mapping to BPEL4WS:~~

- ~~None: this does not map to any BPEL4WS element.~~
- ~~Message: A *receive* will be associated with the message defined with the Message Flow that arrives at the Start Event (see Figure 2).~~



~~Figure 2 Message Flow connected to a Start Event~~

- ~~If there is more than one connected to the Start Event, then a BPEL4WS *pick* will be required to process the messages with a separate *receive* for each message. This~~

means that a single instance of the process will be instantiated when the first message received through the *pick receives* arrives.

Note: The modeler does not need to connect the Message Flows to the Start Event to model this behavior, however. The receipt of the messages could be spelled out through the modeling of the receiving Tasks as graphical objects (see Figure 3) and using a None Start Event.

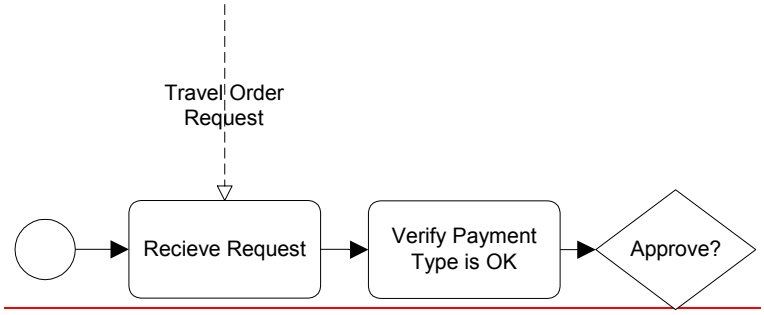


Figure 3 Process Instantiation through Message Receiving Task

- Timer: TBD.
- Rule: TBD.
- Link: this will map to the *receive* element.
- Multiple: this will map to a combination of *receive* elements.

BPML

- If the Start Event has an expression for the assign property, then this will map to a BPML *assign*.

Each type of Start Event Trigger will have a different mapping to BPML:

- None: this does not map to any BPML element.
- Message: A *one-way action* will be associated with the message defined with the Message Flow that arrives at the Start Event (see Figure 2).
  - If there is more than one connected to the Start Event, then a BPML *choice* will be required to process the messages with a separate *one-way action* for each message. This means that a single instance of the process will be instantiated when the first message received through the *choice actions* arrives.

Note: The modeler does not need to connect the Message Flows to the Start Event to model this behavior, however. The receipt of the messages could be spelled out through the modeling of the receiving Tasks as graphical objects (see Figure 3).

- ~~Timer: this will map to a *faults case* that is triggered by a *schedule* element within a *context*.~~
- ~~Rule: TBD.~~
- ~~Link: this will map to the *signal* within an *event* element.~~
- ~~Multiple: this will map to a combination of *action*, *schedule*, *signal*, and TBD elements.~~

Refer to the section entitled “Start Events” on page 147 and the section entitled “Start Events” on page 154 for more information about how the Start Event maps to excution languages.

4.2.2 End

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

The End Event shares the same basic shape of the Start Event and Intermediate Event, a circle, but is drawn with a thick single line (see Figure 4). Text associated with the End Event (e.g., its name) can be placed above or below the shape, in any direction or location, or on the incoming Sequence Flow, depending on the preference of the modeler or modeling tool vendor.

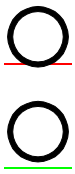


Figure 4 End Event

To continue ~~the~~ discussing how flow proceeds throughout the process, 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 from the Start Events or through forking during the Process must be consumed before the Process has been completed.

Semantics of the End Event include:

- There ~~can~~ MAY be multiple End Events within a single level of a process.
- *This shape is optional*~~—~~: a given Process ~~level (~~level—a top-level Process or an expanded Sub-Process~~)~~ —MAY (is not required ~~to~~ to) have this shape:
  - If there is ~~an End~~ a Start Event, then there ~~has to~~ MUST be at least one ~~Start~~ End Event.
  - If an End Event is used, then there ~~can~~ SHALL NOT be ~~no~~ other flow elements that do not have any outgoing Sequence Flows (of those elements that can generate Sequence Flow)—all other flow objects ~~must~~ MUST be a source of at least one Sequence Flow.

- If the End Event is not used, then all flow objects that do not have any outgoing Sequence Flows (i.e., are not a source of a Sequence Flow) mark the end of the process. However, the process ~~will not~~ SHALL NOT end until all parallel paths have completed.

Note: A BPD may have more than one Process level (i.e., it can include Expanded Sub-Processes). The use of Start and End Events is independent for each level of the diagram.

A Token entering the path-ending flow objects will be consumed when the processing performed by those objects are completed (when the path has completed). When all Tokens for a given instance of the Process are consumed, then the Process will reach a state of being completed. However, a Process may be given attributes to control how Tokens moves back up to a higher-level Process. This is an open issue. Refer to the section entitled “Open Issues” on page 165 for a complete list of the issues open for BPMN.

End Event Results

A BPMN modeler can define the consequence of reaching an End Event. This will be referred to as the End Event Result.

Table 8 displays the types of Results and the graphical marker that will be used for each:

Result	Description	Marker
None	The modeler does not display the type of Event. It is also used for a Sub-Process that end and the flow goes back to its Parent Process.	
Message	A message is sent to a participant at the conclusion of the Process.	
Process Error	This particular End will inform the Process Engine that named Error should be generated. This Error will be caught by an Intermediate Event within the Event Context.	
Compensate	This particular End will inform the Process Engine that a Compensation is necessary. This Compensate identifier will be used by an Intermediate Event when the Process is rolling back.	
Link	A Link is a mechanism for connecting the end (Result) of one Process to the start (Trigger) of another. Typically, these are two Sub-Processes within the same parent Process.	
Multiple	This means that there are multiple consequences of ending the Process. All of them will occur. The attributes of the End Event will define which of the other types of Results apply.	

Result	Description	Marker
None	The modeler does not display the type of Event. It is also used for a Sub-Process that end and the flow goes 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.	
Exception	This type of End will inform the Process Engine that named Error should be generated. This Error will be caught by an Intermediate Event within the Event Context.	
Compensation	This type of End will inform the Process Engine that a Compensation is necessary. This Compensation identifier will be used by an Intermediate Event when the Process is rolling back.	
Link	A Link is a mechanism for connecting the end (Result) of one Process to the start (Trigger) of another. Typically, these are two Sub-Processes within the same parent Process.	
Terminate	This type of End indicates that there is a fatal error and that all activities in the Process should be immediately ended. The Process is ended without compensation or event handling.	

Return	This type of End is for the special circumstances of compensation. When a compensation is triggered, it can be triggered from a downstream Event. The flow will Jump from the compensation source to the compensation target. When the compensation activities have been completed, as indicated by the End Event, then the flow will “return” back to the source so that the process can continue.	
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 8 End Event Types

Attributes

The following are ~~identified~~ attributes of ~~an~~ a End Event, which extends the set of common object elements (see Table 5):

Attribute	Description
Name ?	Name is an optional property that is text description of the Event.
Result: (None | Message | ProcessError | Compensate | Rule | Link | Multiple) : None	Result is a property (default None) that defines the type of result expected for that End.
Message: MessageName	If the Result is a Message, then the name of the Message must be supplied.
ProcessError: ErrorCode	If the Result is a Process Error, then the error code must be supplied.
Compensate: CompensateCode	If the Result is a Compensate, then the compensation code must be supplied.
Link: LinkName	If the Result is a Link, then the name of the Link must be supplied.
Multiple: Trigger + (except Multiple)	If the Result is a Multiple, then a list of the Results must have the appropriate data.
Assign *: Expresssion	Zero or more assignments can be made. Each assignment is an expression.
IncomingSequenceFlow +: SequenceFlowName	One or more incoming Sequence Flows can be idenitified for the End.
FlowCondition: (One | All | Complex) : All	If there is more than one, or if there are more than one End Events, then a Flow Condition must be set. The Flow Condition will apply to all End Events for that level of the Process. A Flow Condition of One means that flow will continue up to the Parent Process when one Token arrives. The process will continue and all other Tokens arriving at will be consumed, but no other Token will proceed up to the Parent Process. A Flow Condition of All means that all Tokens generated at that level of the Process must be consumed before a Token is passed back up to the Parent Process.
Complex: Expression	A complex Flow Condition can be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process data.
PassThrough: (True | False): False	The definition of the PassThrough property is an open issue that will be handled in a later version of the specification. Refer to the section entitled “Open Issues” on page 137 for a complete list of the issues open for BPMN.

Attribute	Description
OutgoingMessageFlow *: MessageFlowName	Zero or more outgoing Message Flows can be idenitified for the End, but the Result type must be Message or Multiple (with Message as one of Result types). For each Message Flow there will be an associated BPEL4WS *reply* or asynchronous *invoke* or a BPML *notification action* to send the message.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the End Event to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the End Event.
Documentation ?	The modeler can add optional text documentation about the End Event.

Attribute	Description
Result: (None | Message | Exception | Compensation | Rule | Link | Terminate | Return | Multiple) : None	Result is an attribute (default None) that defines the type of result expected for that End.
Message: MessageName	If the Result is a Message, then the name of the Message MUST be supplied.
Exception: ErrorCode	If the Result is an Exception, then the error code MAY be supplied.
Compensation: ActivityName; ActivityID	If the Result is a Compensation, then the name of the Activity that needs to be compensated MAY be supplied.The ActivityID of that activiy MUST be supplied.
Link: LinkName	If the Result is a Link, then the name of the Link MUST be supplied.
Multiple: Trigger + (except Multiple)	If the Result is a Multiple, then each Result on the list MUST have the appropriate data as specified for the above attributes.

Table 9 End Event Attributes

Sequence ~~flow~~ Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how they may be source or targets of Sequence Flows.

- ~~An End Event can be a target for a Sequence Flow; there can be multiple incoming Flows. The Flows can come from either alternative or parallel paths. For modeling convenience, each path can connect to a separate End Event object or one End Event can be used.~~
- An End Event MUST be a target for a Sequence Flow.
- An End Event MAY have multiple incoming Flows.

~~Thus~~The Flows MAY come from either alternative or parallel paths. For modeling convenience, ~~the~~ 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 or consumed through an exception handling Intermediate~~ Event. The Process will be in a *running* state until all Tokens are consumed.

- An End Event ~~cannot~~ SHALL NOT be a source for a Sequence Flow; that is, there ~~can~~ SHALL NOT be ~~no~~ outgoing Flows.

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how they may be source or targets of Sequence Flows.

Note: All Message Flows described here 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 ~~cannot~~ MUST NOT be the target for Message Flows; it can have no incoming Message Flows.
- An End Event ~~can~~ MAY be a source for a Message Flow; it can have one or more outgoing Message Flow.
  - ~~However, if there is more than one End Event in the Process, then each of the End Events can have a different outgoing Message Flow.~~

Mapping to Execution Languages

~~The following two sections describe how the use of End Events will map to BPEL4WS and BPML, respectively.~~

~~BPEL4WS~~

- ~~If the End Event has an expression for the assign property, then this will map to a BPEL4WS *assign*.~~

~~Each type of End Event Result will have a different mapping to BPML:~~

- ~~None: this does not map to any BPEL4WS element. However, it marks the end of a path within the Process and will be used to define the boundaries of complex BPEL4WS elements.~~
- ~~Message: A graphically hidden BPEL4WS *reply* will be associated with the message defined with the Message Flow that leaves the End Event (see Figure 5).~~

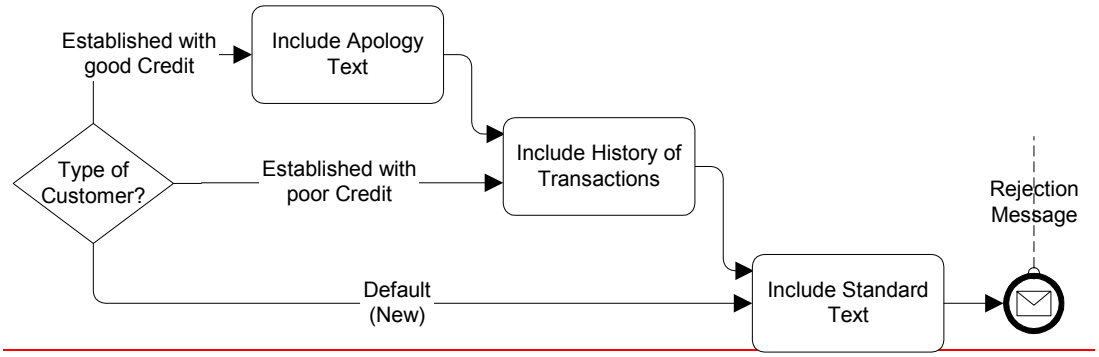


Figure 5 Message Flow leaving an End Event

Note: The modeler does not need to connect the Message Flows from the End Event to model this behavior, however. The sending of the messages could be modeled through the modeling of the sending Tasks as graphical objects (see Figure 6)

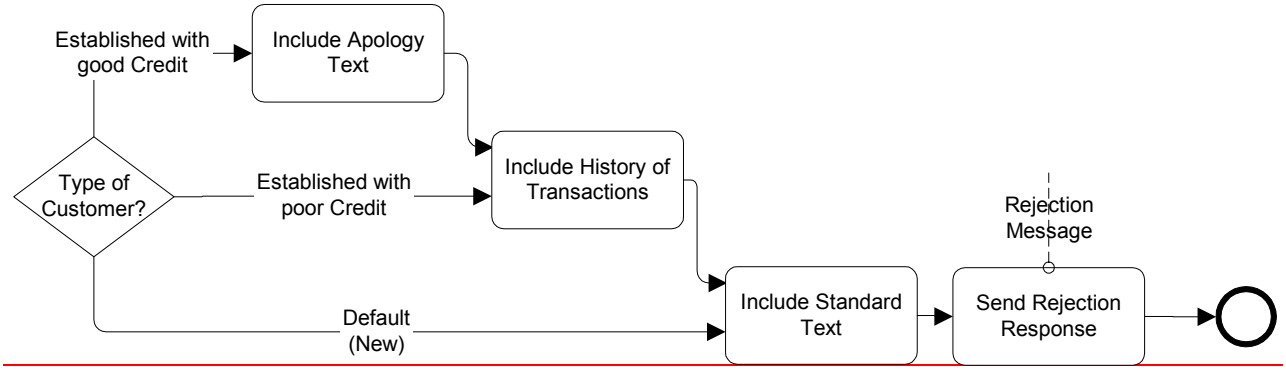


Figure 6 Message Flow from Task that precedes the End Event

- Process Error: this will map to a *throw* element.
- Compensate: this will map to a *compensate* element.
- Link: this will map to the *invoke* element.
- Multiple: this will map to a combination of *invoke*, *throw*, *fault*, and *compensate* elements.

BPML

- If the End Event has an expression for the assign property, then this will map to a BPML *assign*.

Each type of End Event Result will have a different mapping to BPML:

- ~~None: this does not map to any BPML element. However, it marks the end of a path within the Process and will be used to define the boundaries of complex BPML elements.~~
- ~~Message: A *notification action* will be associated with the message defined with the Message Flow that leaves the End Event (see Figure 5).~~

~~**Note**: The modeler does not need to connect the Message Flows from the End Event to model this behavior, however. The sending of the messages could be modeled through the modeling of the sending Tasks as graphical objects (see Figure 6).~~

- ~~Process Error: this will map to a *fault* element.~~
- ~~Compensate: this will map to a *compensate* element.~~
- ~~Link: this will map to the *raise* element.~~
- ~~Multiple: this will map to a combination of *action*, *raise*, *fault*, and *compensate* elements.~~

Refer to the section entitled “End Events” on page 148 and the section entitled “End Events” on page 155 for more information about how the Start Event maps to excution languages.

4.2.3 Intermediate

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 or delays are expected within the Process,
- Disrupt the normal flow through exception handling, or
- Show the extra work required for ~~compensating a transaction~~compensation.

The Intermediate Event shares the same basic shape of the Start Event and End Event, a circle, but is drawn with a thin double line (see Figure 7). Text associated with the Intermediate Event (e.g., its name) can be placed above or below the shape, in any direction or location, or on the outgoing Sequence Flow, depending on the preference of the modeler or modeling tool vendor.

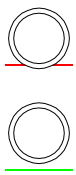


Figure 7 Intermediate Event

One use of Intermediate Events is to represent exception or ~~transaction~~ compensation handling. This will be shown by placing the Intermediate Event on the boundary of a Task or Sub-Process (either collapsed or expanded). Figure 8 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 and 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 and then down.

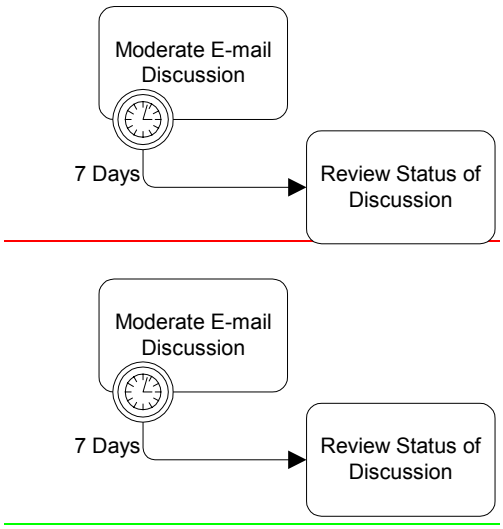


Figure 8 Task with an Intermediate Event attached to its boundary

Intermediate Event Triggers

There are ~~seven~~ eight types of Intermediate Events in BPMN: Message, Timer, ~~Process Error (exception)~~Exception, ~~Compensate~~Compensation, Rule, Link, and Multiple. These Event types indicate the different ways that a Process may be interrupted or delayed after it has started. Each type of Intermediate Event will have a different icon placed in the center of the Intermediate Event shape to distinguish one from another.

Table 10 displays the types of Triggers and the graphical marker that will be used for each:

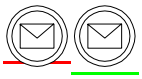
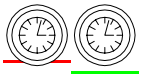
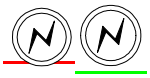
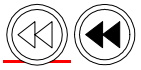
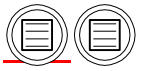
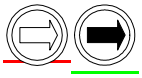
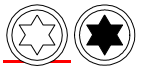
Trigger	Description	Marker
Message	A message arrives from a participant and triggers the Event. This causes the Process to continue if it was waiting for the message, or changes the flow for exception handling.	
Timer	A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that 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.	
~~Process Error~~Exception	This is ~~only~~ used for exception handling~~--both setting~~ and reacting to exceptions. It creates an exception if the Event is part of a normal flow. It reacts to a named ~~Error (e.g.~~exception, ~~thrown from an End Event)~~ or to any ~~error~~ exception if a name is not specified, when attached to the boundary of an activity.	
~~Compensate~~Compensation	This is ~~only~~ used for ~~transaction~~ compensation handling~~--both setting and performing compensation~~. It call for compensation if the Event is part of a normal flow. It reacts to a named ~~Compensate (e.g., thrown from~~ compensation call when attached to the boundary of an ~~End Event)~~activity.	
Rule	This is only used for exception handling. This type of event is triggered when a named Rule becomes true. A Rule is an expression that evaluates some Process data.	
Link	A Link is a mechanism for connecting the end (Result) of one Process to the start (Trigger) of Event-Based Exclusive Decision.	
Multiple	This means that there are multiple ways of triggering the Event. Only one of them will be required. The attributes of the Intermediate Event will define which of the other types of Triggers apply.	

Table 10 Intermediate Event Types

Attributes

The following are ~~identified~~ attributes of an Intermediate Event, which extends the set of common object elements (see Table 5):

Attribute	Description
Name ?	Name is an optional property that is text description of the Event.
Trigger: (Message | Timer | ProcessError | Compensate | Rule | Multiple) : Message	Trigger is a property (default Message) that defines the type of trigger expected for that Intermediate Event.

Attribute	Description
Message: MessageName	If the Trigger is a Message, then the name of the Message must be supplied.
Timer: (Timedate | TimeCycle): Timedate	If the Trigger is a Timer, then a timedate or a timedatecycle must be entered.
ProcessError: (ErrorCode | None): ErrorCode	If the Trigger is a Process Error, then the error code can be supplied. If there is no error code, then any Error will trigger the Event.
Compensate: CompensateCode	If the Trigger is a Compensate, then the compensation code must be supplied.
Rule: RuleName	If the Trigger is a Rule, then an expression must be entered.
Link: LinkName	If the Trigger is a Link, then the name of the Link must be supplied.
Multiple: Trigger + (except Multiple)	If the Trigger is a Multiple, then a list of the Trigger must have the appropriate data.
Interrupt: (True | False): True	Interrupt is a property that has a default of True. The property defines how the Intermediate Event will affect the Event Context if the Intermediate Event is attached to the boundary of an activity. If the property is True, then the Intermediate Event will interrupt the processing of all the activities within the Event Context. The flow would then be diverted through the outgoing Sequence Flow from the Intermediate Event. If the property is False, all processing of all the activities within the Event Context will continue *and* the flow will also be sent through the outgoing Sequence Flow from the Intermediate Event.
Assign *: Expression	Zero or more assignments can be made. Each assignment is an expression.
IncomingSequenceFlow *: SequenceFlowName	Zero or more incoming Sequence Flows can be idenitified for the Intermediate Event.
OutgoingSequenceFlow +: SequenceFlowName	One or more outgoing Sequence Flows can be idenitified for the Intermediate Event.
IncomingMessageFlow *: MessageFlowName	Zero or more incoming Message Flows can be idenitified for the Intermediate Event, but the Trigger type must be Message or Multiple (with Message as one of Trigger types). For a Message Flow there will be an associated BPEL4WS *receive* BPML *action* to receive the message. For multiple Message Flows, there will be BPEL4WS *pick* or BPML *choice* that will receive only one of the messages.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Intermediate Event to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the Intermediate Event.

Attribute	Description
Documentation ?	The modeler can add optional text documentation about the Intermediate Event.

Attribute	Description
Trigger: (Message | Timer | Exception | Compensation | Rule | Multiple) : Message	Trigger is an attribute (default Message) that defines the type of trigger expected for that Intermediate Event.
Message: MessageName	If the Trigger is a Message, then the name of the Message must be supplied.
Timer: (Timedate | TimeCycle): Timedate	If the Trigger is a Timer, then a timedate or a timecycle must be entered.
Exception: (ErrorCode | None): ErrorCode	If the Trigger is an Exception, then the error code MAY be supplied. If there is no error code, then any Error SHALL trigger the Event. If there is an error code, then only an Error that matches the error code SHALL trigger the Event.
Compensation: ActivityName; ActivityID	If the Trigger is a Compensation, then the name of the Activity needs to be compensated MAY be supplied.The ActivityID of that activity MUST be supplied.
Rule: RuleName	If the Trigger is a Rule, then an expression MUST be entered.
Link: LinkName	If the Trigger is a Link, then the name of the Link MUST be supplied.
Multiple: Trigger + (except Multiple)	If the Trigger is a Multiple, then each Trigger on the list MUST have the appropriate data as specified for the above attributes.

Table 11 Intermediate Event Attributes

~~**Note**: BPMN may include a Process Break element in a future version of the Specification. The Process Break would be used in conjunction with a Timer Intermediate Event and highlights the location of a delay in the Process. This is an open issue. Refer to the section entitled "Open Issues" on page 137 for a complete list of the issues open for BPMN.~~

Note: BPMN may include a Process Break element in a future version of the Specification. The Process Break would be used in conjunction with a Timer Intermediate Event and highlights the ~~graphical depiction~~ location of a ~~false Interrupt property has not been defined~~delay in the Process. This is an open issue. Refer to the section entitled "Open Issues" on page 165 for a complete list of the issues open for BPMN.

Sequence ~~flow~~ Flow Connections

Refer to the section entitled "Sequence Flow Rules" on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- If the Intermediate Event is not attached to the boundary of an activity, then it MAY be a target for a Sequence Flow; it can have one incoming Flow.
  - Intermediate Event of the following types MAY be a target for a Sequence Flow; that is, be within normal flow: Message, Timer, Exception, and Compensation.
- If the Intermediate Event is attached to the boundary of an activity, then it MAY NOT be a target for a Sequence Flow; it cannot have an incoming Flow.
  - Intermediate Event of the following types MAY be attached to the boundary of an activity and generate exception flow: Message, Timer, Exception, Rule, and Multiple.
  - Intermediate Event of the following types MAY be attached to the boundary of an activity and generate compensation flow: Compensation.
- An Intermediate Event ~~can~~ MAY be a ~~target~~ source for a Sequence Flow; it can have one ~~incoming~~ outgoing Flow.
- ~~An Intermediate Event can be a source for a Sequence Flow; it can have one outgoing Flow.~~

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

Note: All Message Flows described here 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 ~~can~~ MAY be the target for Message Flows; it can have one incoming Message Flows.
- An Intermediate Event ~~cannot~~ MAY NOT be a source for a Message Flow; it can have no outgoing Message Flows.

Mapping to Execution Languages

~~The Mapping to Execution Languages will depend on the type of Intermediate Event. Each of the seven types will be mapped differently.~~

~~**BPEL4WS**~~

- ~~If the Intermediate Event has an expression for the assign property, then this will map to a BPEL4WS *assign*.~~

~~Each type of Intermediate Event will have a different mapping to BPML:~~

- ~~Message:~~
  - ~~If the Intermediate Event follows a Decision (e.g., is part of a *pick*): this will map to an *onMessage* element within a *pick*.~~

- If the Intermediate Event is within the normal flow of the Process (but does not follow a Decision): this will map to a *receive*.
- If the Intermediate Event is attached to the boundary of an activity: this will map to an *onMessage* element within a *scope*.

- Timer:
  - If the Intermediate Event follows a Decision (e.g., is part of a *pick*): this will map to an *onAlarm* element within a *pick*.
  - If the Intermediate Event is within the normal flow of the Process (but does not follow a Decision): this will map to a *wait*.
  - If the Intermediate Event is attached to the boundary of an activity: this will map to a *wait* element, followed by a *throw*. A *scope* is also created that has a *catch* to correspond with the *throw*.
- Process Error: this will map to a *catch* element within a *scope*
- Compensate (must be attached to the boundary of an activity): this will map to an *compensationHandler* element within a *scope*.
- Rule (must be attached to the boundary of an activity): TBD.
- Link (must follow a Decision): this will map to the *onMessage* element of a *pick*.
- Multiple (must be attached to the boundary of an activity): this will map to a combination of *onMessage*, *onAlarm*, *compensationHandler*, *onSignal*, *throw*, *catch*, and *wait* elements within a *context*.

BPML

- If the Intermediate Event has an expression for the assign property, then this will map to a BPML *assign*.

Each type of Intermediate Event will have a different mapping to BPML:

- Message:
  - If the Intermediate Event follows a Decision (e.g., is part of a *choice*): this will map to an *action* within an *event* element within a *choice*.
  - If the Intermediate Event is within the normal flow of the Process (but does not follow a Decision): this will map to an *action* (that expects a message).
  - If the Intermediate Event is attached to the boundary of an activity: this will map to an *action* within an *event* element within an *exception* element within a *context*.
- Timer:
  - If the Intermediate Event follows a Decision (e.g., is part of a *choice*): this will map to a *delay* within an *event* element within a *choice*.
  - If the Intermediate Event is within the normal flow of the Process (but does not follow a Decision): this will map to a *delay*.
  - If the Intermediate Event is attached to the boundary of an activity: this will map to an *schedule* element within a *context* that will create a *fault code* that will be captured by a *faults case* within the *context*.

- ~~Process Error:~~
  - ~~If the Intermediate Event is attached to the boundary of an activity: this will map to an *faults case* within a *context*.~~
- ~~Compensate (must be attached to the boundary of an activity): this will map to an *compensation* process within a *context*.~~
- ~~Rule (must be attached to the boundary of an activity): TBD.~~
- ~~Link (must follow a Decision): this will map to a *signal* within an *event* element of a *choice*.~~
- ~~Multiple (must be attached to the boundary of an activity): this will map to a combination of *actions*, *events*, *faults*, *schedules*, *exceptions*, and *contexts* elements within a *context*.~~

Refer to the section entitled “Intermediate Events” on page 149 and the section entitled “Intermediate Events” on page 155 for more information about how the Start Event maps to excution languages.

4.3 Activities

An activity is work that is performed within a business process. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Process Model are: Process, Sub-Process, and Task. The following sections will detail how these activities are modeled with BPMN.

4.3.1 Processes

A **Process** is an activity performed within a company or organization. In BPMN a Process is depicted as a network of flow objects, which are a set of other activities and the controls that sequence them. The concept of process is intrinsically hierarchical. Processes may be defined at any level from enterprise-wide processes to processes performed by a single person. Low-level processes may be grouped together to achieve a common business goal.

A BPD may contain more than one Process--not including Sub-Processes. Each Process would be contained within a Pool (refer to the section entitled “Pool” on page 100) and would be independent in terms of Sequence Flow, but could have Message Flow connecting them.

Attributes

The following are ~~identified~~ attributes of a Process, which extends the set of common object elements (see Table 5):

Attribute	Description
Name	Name is a text description of the Process.
Nested:(True | False): False	The Nested property defines whether or not the Process is nested within another Process (thus sharing the Parent Process properties).

Attribute	Description
ParentProcess: ProcessName	If Nested, then the Parent Process must be identified.
Property *:	Modeler-defined Properties can be added to a Process. These Properties are “local” to the Process. All Tasks, Sub-Process objects, and Sub-Processes that are nested have access to these Properties. The fully delineated name of these properties are “<process name>.<property name>” (e.g., “Add Customer.Customer Name”). If a process is nested within another Process, then the fully delineated name would also be preceded by the Parent Process name for as many Parents there are until the top level Process.
Name:	Each Property has a Name (e.g., name=”Customer Name”).
Type:	Each Property has a Type (e.g., type=”String”).
AdHoc: (True | False): False	AdHoc is a Boolean property, 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 by a process engine, they are completely controlled by the performers of the activities. The Process Engine may be able to track the actual instances on the activities within.
CompletionCondition: Expression	If the Process is Ad Hoc, then a Completion Condition must be included, which defines the conditions when the Process will end. The Ad Hoc marker will be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.
Assign *: Expression	Zero or more assignments can be made. Each assignment is an expression.
AssignTime: (Start | End): Start	For each assignment the modeler can specify whether the assignment will take place at the start or end of the Process.
IncomingMessageFlow *: MessageFlowName	Zero or more incoming Message Flows can be idenitified for the Process.
Target: (Boundary | Internal): Boundary	Each Message Flow must be associated with the Process itself (Pool Boundary) or with flow objects that the Process contains (Internal). To create a mapping to BPEL4WS or BPML, all the Message Flows should be connected to objects within the Process.
OutgoingMessageFlow *: MessageFlowName	Zero or more outgoing Message Flows can be idenitified for the Process.
Source: (Boundary | Internal): Boundary	Each Message Flow must be associated with the Process itself (Pool Boundary) or with flow objects that the Sub-Process contains (Internal). To create a mapping to BPEL4WS or BPML, all the Message Flows should be connected to objects within the Process.

Attribute	Description
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Process to identify its location. There is a one-to-one relationship between a Process and a Pool.
Partner *: PartnerName	Zero or more Partner's can be identified for the Process. A Partner is also equivalent to the other Pools of the diagram. Thus, all the other Pool Names in the diagram will be listed in as Partners for the current Process. Additional Partners that are not shown as Pools can also be identified. This will map to the *partner* element of BPEL4WS.
Association *	Zero or more Associations can be associated with the Process.
Documentation ?	The modeler can add optional text documentation about the Process.

Attribute	Description
Property *:	Modeler-defined Properties MAY be added to a Process. These Properties are "local" to the Process. All Tasks, Sub-Process objects, and Sub-Processes that are embedded SHALL have access to these Properties. The fully delineated name of these properties are "<process name>.<property name>" (e.g., "Add Customer.Customer Name"). If a process is embedded within another Process, then the fully delineated name SHALL also be preceded by the Parent Process name for as many Parents there are until the top level Process.
Name:	Each Property has a Name (e.g., name="Customer Name").
Type:	Each Property has a Type (e.g., type="String").
AdHoc: (True | False): False	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 by a process engine, they are completely controlled by the performers of the activities. The Process Engine may be able to track the actual instances on the activities within.
CompletionCondition: Expression	If the Process is Ad Hoc, then a Completion Condition MUST be included, which defines the conditions when the Process will end. The Ad Hoc marker SHALL be placed at the bottom center of the Process or the Sub-Process shape for Ad Hoc Processes.
PassThrough: (True | False): False ????	The definition of the PassThrough attribute is an open issue that will be handled in a later version of the specification. Refer to the section entitled "Open Issues" on page 165 for a complete list of the issues open for BPMN.
AssignTime: (Start | End): Start	Each Assignment Expression will have AssignTime. A value of Start means that the assignment SHALL occur at the start of the Process. A value of End means that the assignment SHALL occur at the end of the Process.

Table 12 Process Attributes

4.3.2 Sub-Process

A **Sub-Process** is a compound activity in that it has detail that is defined as a flow of other activities. A Sub-Process is a graphical object within a Process Flow, but it also references another Process (either ~~nested~~ embedded or independent). A Sub-Process shares the same shape as the Task, which is a rectangle that has rounded corners. The Sub-Process can be in a collapsed view that hides its details (see Figure 9) or a Sub-Process can be in an expanded view that shows its details within the view of the Process in which it is contained (see Figure 10). In the collapsed form, the Sub-Process object uses a marker to distinguish it as a Sub-Process, rather than a Task. The marker is a small square with a plus sign (+) inside. The square is positioned at the bottom center of the shape. Text associated with the Sub-Process (e.g., its name) can be placed inside the shape, or above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

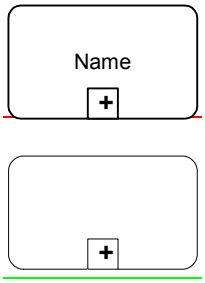


Figure 9 Collapsed Sub-Process

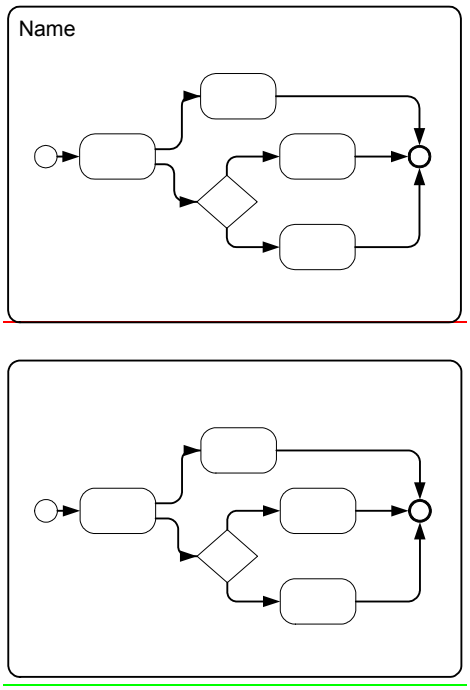


Figure 10 Expanded Sub-Process

Expanded Sub-Process 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 (Refer to the

section entitled “Exception Flow” on page 107 for more details). Compensations can be handled the similarly (Refer to the section entitled “Transaction Compensation Flow” on page 101 for more details).

Expanded Sub-Process may be used as a mechanism for showing a group of parallel activities in a less-cluttered, more compact way. In Figure 11, 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 Even 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.

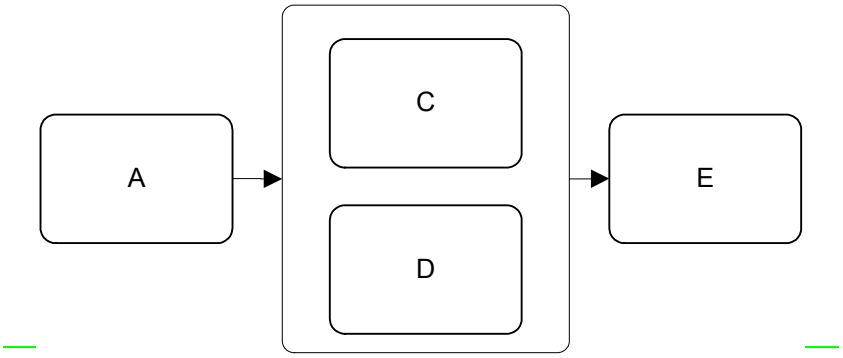


Figure 11 Expanded Sub-Process used as a “parallel box”

BPMN specifies ~~three~~ four types of markers for Sub-Processes. The (Collapsed) Sub-Process Marker, seen in Figure 9, can be combined with ~~two~~ three other markers: a Loop Marker or a Parallel Marker and an Ad Hoc Marker. A Sub-Process may have one or ~~both~~ two of these other markers. All the markers that are present will be grouped and the whole group will be centered at the bottom of the shape (see Figure 12).

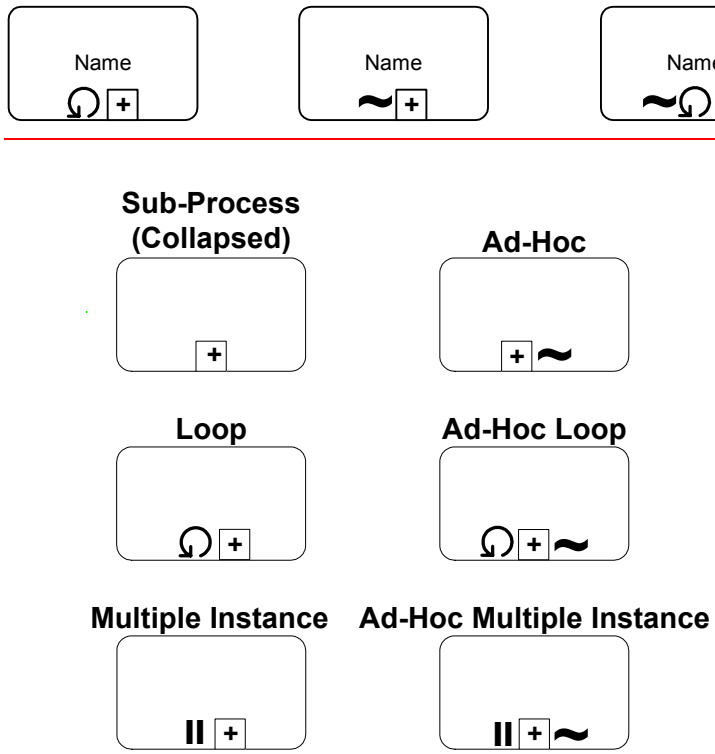


Figure 12 ~~Collapse~~ Collapsed Sub-Process Marker Combinations

~~**Note**: The positioning of the Loop and Ad-Hoc Markers is subject to change in a later release of the BPMN specification. This is an open issue. Refer to the section entitled "Open Issues" on page 137 for a complete list of the issues open for BPMN.~~

~~**Note**: An additional graphical marker may be included in BPMN for parallel ForEach types of loops. This is an open issue. Refer to the section entitled "Open Issues" on page 137 for a complete list of the issues open for BPMN.~~

Attributes

The following are ~~identified~~ attributes of a Sub-Process, which extends the set of common object elements (see Table 5):

Attributes	Description
Name	Name is a text description of the Sub-Process.
SubProcessType: (Nested | Independent): Nested	SubProcessType is a property that defines whether the Sub-Process details are embedded within the higher level Process (nested) or refers to another, re-usable Process. The default is Nested.
Process: ProcessName	If the type is Independent, then the name of the referenced Process must be included.

Attributes	Description
InputMap +: Expression	For Independent, multiple input mappings can be made between Parent Process properties and the properties of the referenced Process. These mappings are in the form of an expression (although a modeling tool can present this to a modeler in any number of ways).
OutputMap +: Expression	For Independent, multiple output mappings can be made between Parent Process properties and the properties of the referenced Process. These mappings are in the form of an expression (although a modeling tool can present this to a modeler in any number of ways).
Property *	Modeler-defined Properties can be added to a Sub.Process. These Properties are “local” to the Sub-Process object—not the Process that the Sub-Process object represents. These Properties are only for use within the processing of the Sub-Process object. The fully delineated name of these properties are “<process name>.<sub-process name>.<property name>” (e.g., “Add Customer.Review Credit.Status”).
Name	Each Property has a Name (e.g., name=”Customer Name”).
Type	Each Property has a Type (e.g., type=”Text”).
Transaction: (True | False): False	Transaction is a Boolean property, which has a default of False. This value automatically becomes True when a Compensate Intermediate Event is attached to the boundary of the Sub-Process.
Compensate: Intermediate Event	The Compensate property lists the name of the Intermediate Event.
EventContext: (True | False): False	EventContext is a Boolean property, which has a default of False. This value automatically becomes True when one or more a Timer, Message, or Process Error Intermediate Events is attached to the boundary of the Sub-Process.
Exception: Intermediate Event	The Exception property lists the names of the Intermediate Events.
LoopType: (None | Standard | ForEach) : None	LoopType is a property and is by default None, but can be set to Standard or ForEach, which means that the Loop marker will be placed at the bottom center of the Sub-Process shape. ForEach Loops require an expression, which specifies the number of instances.
LoopCondition: Expression	Standard Loops required an expression to be evaluated, plus the timing when the expression will be evaluated.
Counter: Number	The Counter property is used at runtime to count the number of loops.

Attributes	Description
Maximum: Number	The Maximum property is a simple way to add a cap to the number of loops. This gets added to the expression when mapped to BPEL4WS or BPML.
EvaluateCondition: (Before | After) : After	Standard Loops expressions evaluated Before the Sub-Process begins are *while* loops and expressions evaluated After the Sub-Process finishes are *while* loops for BPEL4WS and *until* loops for BPML.
Timing: (Serial | Parallel) : Serial	The Timing property defines whether the ForEach instances will be performed serially or in parallel. A parallel ForEach is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use.
LoopFlowCondition: (One | All | Complex): All	The LoopFlowCondition, applied only to Parallel ForEach loops, acts the similarly to the FlowCondition for the Sub-Process. A Loop Flow Condition of One means that the Token will continue past the Sub-Process after only on of the Sub-Process instances has completed. The Sub-Process will continue its other instances, but no other Tokens will be passed from the Sub-Process. A Loop Flow Condition of All means that all Sub-Process instances must be completed before the Token can move from the Sub-Process.
Complex: Expression	A complex Loop Flow Condition can be set by the modeler. This will consist of an expression that can reference Process data. The expression will determine the Token will continue past the Sub-Process.
Assign *: Expression	Zero or more assignments can be made. Each assignment is an expression.
AssignTime: (Start | End): Start	For each assignment the modeler can specify whether the assignment will take place at the start or end of the Sub-Process.
IncomingSequenceFlow +: SequenceFlowName	One or more incoming Sequence Flows can be idenitified for the Sub-Process.
FlowCondition: (One | All | Complex): All	If there is more than one, then a Flow Condition must be set. A Flow Condition of One means that the Sub-Process will be started when one Token arrives on any of the Flows. The process will continue and all other Tokens arriving at the Sub-Process will be consumed. A Flow Condition of All means that a Token must arrive from all incoming Flows before the Sub-Process can start.
Complex: Expression	A complex Flow Condition can be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process data. The expression will determine when the Sub-Process will start.
OutgoingSequenceFlow +: SequenceFlowName	One or more outgoing Sequence Flows can be idenitified for the Sub-Process.

Attributes	Description
IncomingMessageFlow *: MessageFlowName	Zero or more incoming Message Flows can be idenitified for the Sub-Process.
Target: (Boundary | Internal): Boundary	Each Message Flow must be associated with the Sub-Process itself (Boundary) or with flow objects that the Sub-Process contains (Internal). For a Boundary Message Flow, it is equivalent to connecting the Message Flows to the Start Event of the Sub-Process.
OutgoingMessageFlow *: MessageFlowName	Zero or more outgoing Message Flows can be idenitified for the Sub-Process.
Source: (Boundary | Internal): Boundary	Each Message Flow must be associated with the Sub-Process itself (Boundary) or with flow objects that the Sub-Process contains (Internal). For a Boundary Message Flow, it is equivalent to connecting the Message Flows to the End Event of the Sub-Process.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Sub-Process to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the Sub-Process.
Documentation ?	The modeler can add optional text documentation about the Sub-Process.

Attributes	Description
Type: (Embedded | Independent): Embedded	Type is an attribute that defines whether the Sub-Process details are embedded within the higher level Process or refers to another, re-usable Process. The default is Embedded.
Process: ProcessName	If the type is Independent, then the name of the referenced Process MUST be included.
InputMap +: Expression	For Independent, multiple input mappings MAY be made between Parent Process properties and the properties of the referenced Process. These mappings are in the form of an expression (although a modeling tool can present this to a modeler in any number of ways).
OutputMap +: Expression	For Independent, multiple output mappings MAY be made between Parent Process properties and the properties of the referenced Process. These mappings are in the form of an expression (although a modeling tool can present this to a modeler in any number of ways).
Property *	Modeler-defined Properties MAY be added to a Sub.Process. These Properties are “local” to the Sub-Process object—not the Process that the Sub-Process object represents. These Properties are only for use within the processing of the Sub-Process object. The fully delineated name of these properties are “<process name>.<sub-process name>.<property name>” (e.g., “Add Customer.Review Credit.Status”).

Attributes	Description
Name: String	Each Property has a Name (e.g., name="Customer Name").
Type	Each Property has a Type (e.g., type="Text").
Transaction: (True | False): False	Transaction is a Boolean attribute, which has a default of False.
LoopType: (None | Standard | ForEach) : None	LoopType is an attribute and is by default None, but MAY be set to Standard or ForEach. If so, the Loop marker SHALL be placed at the bottom center of the Sub-Process shape.
LoopCondition: Expression	Standard and ForEach Loops MUST have an expression to be evaluated, plus the timing when the expression SHALL be evaluated.
Counter: Number	The Counter attribute is used at runtime to count the number of loops.
Maximum ?: Number	For Standard Loops, 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 when mapped to BPEL4WS or BPML.
EvaluateCondition: (Before | After) : After	Standard Loop expressions evaluated Before the Sub-Process begins are *while* loops and expressions evaluated After the Sub-Process finishes are *while* loops for BPEL4WS and *until* loops for BPML.
Timing: (Serial | Parallel) : Serial	The Timing attribute defines whether the ForEach instances will be performed serially or in parallel. A serial ForEach is a more traditional loop. A parallel ForEach 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 Sub-Process shape.
LoopFlowCondition: (One | All | Complex): All	This applies only to Parallel ForEach Loops. It is equivalent to using a Gateway to control the flow past a set of parallel paths. A LoopFlowCondition of One means that the Token SHALL continue past the Sub-Process after only one of the Sub-Process instances has completed. The Sub-Process SHALL continue its other instances, but additional Tokens SHALL NOT be passed from the Sub-Process. A LoopFlowCondition of All means that all Sub-Process instances MUST be completed before the Token can move from the Sub-Process.
Complex ?: Expression	A complex Loop Flow Condition MAY be set by the modeler. This will consist of an expression that MAY reference Process data. The expression SHALL determine when and how many Tokens will continue past the Sub-Process.
AssignTime: (Start | End): Start	Each Assignment Expression will have AssignTime. A value of Start means that the assignment SHALL occur at the start of the Sub-Process. A value of End means that the assignment SHALL occur at the end of the Sub-Process.

Table 13 Sub-Process Attributes

Sequence ~~flow~~ Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Sub-Process ~~can~~ MAY be a target for a Sequence Flow; it can have multiple incoming Flows. ~~An incoming~~ Incoming Flow ~~can~~ MAY be from an alternative path and/or a parallel ~~path. The Flow Condition will determine when the Sub-Process will start~~paths.

  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. If the flow needs to be controlled, then the flow should converge on a Gateway that preceeds the Sub-Process (Refer to the section entitled “Gateways” on page 76 for more information on Gateways).

  - If the Sub-Process does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Sub-Process ~~will~~ MUST be instantiated when the process is instantiated.

- A Sub-Process ~~can~~ MAY be a source for a Sequence Flow; it can have multiple outgoing Flows. If there are multiple outgoing Sequence Flows, then this means that a separate parallel path is being created for each Flow.

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 (AND-Split) and the number of Tokens in the group

- 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 ~~will~~ MUST be completed.

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

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

Mapping to Execution Languages

The following two sections describe how the use of Sub-Processes will map to BPEL4WS and BPML, respectively.

BPEL4WS

There are four possibilities, depending on the Message Flows that attach to the Sub-Process boundary:

- A Sub-Process that has no Message Flows attached to its boundary will map to the BPEL4WS *invoke*. This will invoke another web service, which is another *process*.
- A Sub-Process that has an incoming Message Flow attached to its boundary will map to a BPEL4WS *receive* followed by a BPEL4WS *invoke*.
- A Sub-Process that has an outgoing Message Flow attached to its boundary will map to the BPEL4WS *invoke* followed by a BPEL4WS *reply*.
- A Sub-Process that has both an incoming and an outgoing Message Flow attached to its boundary will map to a BPEL4WS *receive* followed by a BPEL4WS *invoke* followed by a BPEL4WS *reply*.

Sub-Process properties will map as follows:

- For a Reference Sub-Process type the modeler will have to create the referenced Process independently (with a different name) and then assign the Process to the Sub-Process object. The referenced *process* will be called with the BPEL4WS *invoke*.
  - InputMap will be mapped to the parameter passing elements of the *call*.
  - OutputMap will be mapped to the parameter passing elements of the *call*.
- The mapping for the Transaction property is TBD.
- If the LoopType is Standard then the Sub-Process will be wrapped by a BPEL4WS *while* or *until*.
  - A Before EvaluateCondition will map to the BPEL4WS *while*.
  - An After EvaluateCondition will map to the BPEL4WS *while*. However, to ensure that the Sub-Process is performed at least once, the *activity(s)* appropriate for the Sub-Process Type will be performed first in a *sequence*, which includes the *while*
  - Any value in Maximum will be appended to the LoopCondition. For example with a LoopCondition of "x < 0" and Maximum of 5 (loops), the final expression would be "(x < 0) and (<Sub-ProcessName>.Counter <= 5)." An BPEL4WS *assign* will be used to update the Counter property.
- If the LoopType is ForEach then the TBD.

Editor's Note: We have not determined how the Ad Hoc Sub-Process will be mapped to BPEL4WS.

BPML

There are four possibilities, depending on the Message Flows that attach to the Sub-Process boundary:

- A Sub-Process that has no Message Flows attached to its boundary will map to the BPML *call*.
- A Sub-Process that has an incoming Message Flow attached to its boundary will map to a BPML *one-way action* followed by a BPML *call*.
- A Sub-Process that has an outgoing Message Flow attached to its boundary will map to the BPML *call* followed by a BPML *one-way action*.
- A Sub-Process that has both an incoming and an outgoing Message Flow attached to its boundary will map to the BPML *request-response action*.
  - If the Sub-Process type is Independent, then a BPML *call* will be used within the *action*.
  - If the Sub-Process type is Nested, then the activities within the Sub-Process will be mapped as appropriate and then inserted within the *action*.

Sub-Process properties will map as follows:

- For a Reference Sub-Process type the modeler will have to create the referenced Process independently (with a different name) and then assign the Process to the Sub-Process object. The referenced *process* will be called with the BPML *call*.
  - InputMap will be mapped to the parameter passing elements of the *call*.
  - OutputMap will be mapped to the parameter passing elements of the *call*.
- The mapping for the Transaction property is TBD.
- If the LoopType is Standard then the Sub-Process will be wrapped by a BPML *while* or *until*.
  - A Before EvaluateCondition will map to the BPML *while*.
  - An After EvaluateCondition will map to the BPML *until*.
  - Any value in Maximum will be appended to the LoopCondition. For example with a LoopCondition of "x < 0" and Maximum of 5 (loops), the final expression would be "(x < 0) and (<Sub-ProcessName>.Counter <= 5)." An BPML *assign* will be used to update the Counter property.
- If the LoopType is ForEach then the Sub-Process will be wrapped by a BPML *foreach*.
  - If the Time is Parallel, then the Sub-Process will be accessed through BPML *spawn* for each instance and then a BPML *synch* will synchronize them.
  - Mapping the LoopFlowCondition TBD.

Editor's Note: We have not determined how the Ad-Hoc Sub-Process will be mapped to BPML.

Refer to the section entitled “Sub-Processes” on page 150 and the section entitled “Sub-Processes” on page 156 for more information about how the Start Event maps to excution languages.

4.3.3 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. Generally, an end-user and/or an application 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 (see Figure 13). Text associated with the Task (e.g., its name) can be placed above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

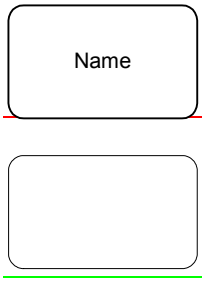


Figure 13 A Task Object

BPMN specifies two types of markers for Task. The (Collapsed) Sub-Process Marker, seen in Figure 13, can be combined with three other markers: a Loop Marker or a Parallel Marker and an Ad Hoc Marker. A Sub-Process may have one or two of these markers. All the markers that are present will be grouped and the whole group will be centered at the bottom of the shape (see Figure 14).

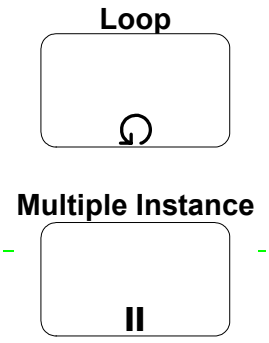


Figure 14 Task Markers

Attributes

The following are ~~identified~~ attributes of a Task, which extends the set of common object elements (see Table 5):

Attributes	Description
Name	Name is a text description of the Task.

Attributes	Description
Type (Send | Receive | Service | User): Service	Type is a property that has a default of Service, but can be set to Send, Receive, or User. The type of Task will depend on the Message Flows to and/or from the Task, if Message Flows are used. If there is only an incoming Sequence Flow, then the type must be Receive. If there is only an outgoing Sequence Flow, then the type must be Send. If there is both an incoming and outgoing Sequence Flow, then the type must be Service. User Tasks are necessary to create an asynchronous mechanism of notification and response to handle the long-lived nature of User Tasks. Since a BPML *solicit-response action* is synchronous, this means that a set of BPML elements will be required to create an asynchronous situation that handles complexities of the interactions between a User and a BPM Engine (this is detailed below in the section on “Mapping to Execution Languages”).
(Receive) **Instantiate** (True | False): False	Receive Tasks can be defined as the instantiation mechanism for the Process with the Instantiate property. This property can only be set to true if the Task is the first activity after the Start Event or a starting activity if there is no Start Event.
Property *	Modeler-defined Properties can be added to a Task. These Properties are “local” to the Task object. These Properties are only for use within the processing of the Task object. The fully delineated name of these properties are “<process name>.<task name>.<property name>” (e.g., “Add Customer.Review Credit Report.Score”).
Name	Each Property has a Name (e.g., name=”Customer Name”).
Type	Each Property has a Type (e.g., type=”Text”).
Input *: Attribute	Input is an optional property that defines which of the Parent Process attributes are used as either an input for or an output from the Task.
Output *: Attribute	Output is an optional property that defines which of the Parent Process attributes are used as either an input for or an output from the Task.
Transaction: (True | False): False	Transaction is a Boolean property, which has a default of False. This value automatically becomes True when a Compensate Intermediate Event is attached to the boundary of the Task.
Compensate: Intermediate Event	The Compensate property lists the name of the Intermediate Event.

Attributes	Description
EventContext: (True | False): False	EventContext is a Boolean property, which has a default of False. This value automatically becomes True when one or more a Timer, Message, or Process Error Intermediate Events is attached to the boundary of the Task.
Exception: Intermediate Event	The Exception property lists the names of the Intermediate Events.
LoopType: (None | Standard | ForEach) : None	LoopType is a property and is by default None, but can be set to Standard or ForEach, which means that the Loop marker will be placed at the bottom center of the Task shape. ForEach Loops require an expression, which specifies the number of instances.
LoopCondition: Expression	Standard Loops required an expression to be evaluated, plus the timing when the expression will be evaluated.
Counter: Number	The Counter property is used at runtime to count the number of loops.
Maximum: Number	The Maximum property is a simple way to add a cap to the number of loops. This gets added to the expression when mapped to BPEL4WS or BPML.
EvaluateCondition: (Before | After) : After	Standard Loops expressions evaluated Before the Sub-Process begins are *while* loops and expressions evaluated After the Task finishes are *while* loops for BPEL4WS and *until* loops for BPML.
Timing: (Serial | Parallel) : Serial	The Timing property defines whether the ForEach instances will be performed serially or in parallel. A parallel ForEach is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use.
LoopFlowCondition: (One | All | Complex): All	The LoopFlowCondition, applied only to Parallel ForEach loops, acts the similarly to the FlowCondition for the Sub-Process. A Loop Flow Condition of One means that the Token will continue past the Task after only on of the Sub-Process instances has completed. The Task will continue its other instances, but no other Tokens will be passed from the Task. A Loop Flow Condition of All means that all Task instances must be completed before the Token can move from the Task.
Complex: Expression	A complex Loop Flow Condition can be set by the modeler. This will consist of an expression that can reference Process data. The expression will determine the Token will continue past the Task.
Assign *: Expression	Zero or more assignments can be made. Each assignment is an expression.
AssignTime: (Start | End): Start	For each assignment the modeler can specify whether the assignment will take place at the start or end of the Task.

Attributes	Description
IncomingSequenceFlow +: SequenceFlowName	One or more incoming Sequence Flows can be idenitified for the Task.
FlowCondition: (One | All | Complex): All	If there is more than one, then a Flow Condition must be set. A Flow Condition of One means that the Task will be started when one Token arrives on any of the Flows. The process will continue and all other Tokens arriving at the Task will be consumed. A Flow Condition of All means that a Token must arrive from all incoming Flows before the Task can start.
Complex: Expression	A complex Flow Condition can be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process data. The expression will determine when the Task will start.
OutgoingSequenceFlow +: SequenceFlowName	One or more outgoing Sequence Flows can be idenitified for the Task.
IncomingMessageFlow *: MessageFlowName	Zero or more incoming Message Flows can be idenitified for the Task.
OutgoingMessageFlow *: MessageFlowName	Zero or more outgoing Message Flows can be idenitified for the Task.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Task to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the Task.
Documentation ?	The modeler can add optional text documentation about the Task.

Attributes	Description
Type (Service | Receive | Send | User | Abstract | None): Service	Type is an attribute that has a default of Service, but MAY be set to Send, Receive, User, or None. The type of Task will depend on the Message Flows to and/or from the Task, if Message Flows are used. If there is only an incoming Sequence Flow, then the type MUST be Receive. If there is only an outgoing Sequence Flow, then the type MUST be Send. If there is both an incoming and outgoing Sequence Flow, then the type MUST be Service. User Tasks are necessary to create an asynchronous mechanism of notification and response to handle the long-lived nature of User Tasks. Since a BPML *solicit-response action* is synchronous, this means that a set of BPML elements will be required to create an asynchronous situation that handles complexities of the interactions between a User and a BPM Engine (this is detailed below in the section on “Mapping to Execution Languages”). <Here?>

Attributes	Description
(Receive) **Instantiate** (True | False): False	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 activity if there is no Start Event. Multiple Tasks MAY have this attribute set to True.
(Abstract) **AbstractType** (MEP1 | MEP2 | MEP3 | MEP4 | MEP5 | MEP6 | MEP7):	Abstract Tasks are used exclusively in Pools of type Interface or Collaboration.
Property *	Modeler-defined Properties MAY be added to a Task. These Properties are “local” to the Task object. These Properties are only for use within the processing of the Task object. The fully delineated name of these properties are “<process name>.<task name>.<property name>” (e.g., “Add Customer.Review Credit Report.Score”).
Name:Text	Each Property has a Name (e.g., name=”Customer Name”).
Type	Each Property has a Type (e.g., type=”Text”).
Input *: Attribute	Input is an optional attribute that defines which of the Parent Process attributes are used as either an input for or an output from the Task.
Output *: Attribute	Output is an optional attribute that defines which of the Parent Process attributes are used as either an input for or an output from the Task.
Transaction: (True | False): False	Transaction is a Boolean attribute, which has a default of False. This value SHALL be True when a Compensation Intermediate Event is attached to the boundary of the Task.
LoopType: (None | Standard | ForEach) : None	LoopType is an attribute and is by default None, but MAY be set to Standard or ForEach. If so the Loop marker SHALL be placed at the bottom center of the Task shape.
LoopCondition: Expression	Standard and ForEach Loops MUST have an expression to be evaluated, plus the timing when the expression will be evaluated.
Counter: Number	The Counter attribute SHALL be used at runtime to count the number of loops.
Maximum ?: Number	For Standard Loops, the Maximum attribute is a simple way to add a cap to the number of loops. This SHALL be added to the expression when mapped to BPEL4WS or BPML.
EvaluateCondition: (Before | After) : After	Standard Loop expressions evaluated Before the Sub-Process begins are *while* loops and expressions evaluated After the Task finishes are *while* loops for BPEL4WS and *until* loops for BPML.
Timing: (Serial | Parallel) : Serial	The Timing attribute defines whether the ForEach instances will be performed serially or in parallel. A parallel ForEach is equivalent to multi-instance specifications that other notations, such as UML Activity Diagrams use.

Attributes	Description
LoopFlowCondition: (One | All | Complex): All	This applies only to Parallel ForEach Loops. It is equivalent to using a Gateway to control the flow past a set of parallel paths. A LoopFlowCondition of One means that the Token SHALL continue past the Task after only one of the Task instances has completed. The Task will continue its other instances, but additional Tokens SHALL NOT be passed from the Task. A LoopFlowCondition of All means that all Task instances MUST be completed before the Token can move from the Task. <here>
Complex ?: Expression	A complex Loop Flow Condition MAY be set by the modeler. This will consist of an expression that can reference Process data. The expression SHALL determine when and how many Tokens will continue past the Task.
AssignTime: (Start | End): Start	Each Assignment Expression will have AssignTime. A value of Start means that the assignment SHALL occur at the start of the Task. A value of End means that the assignment SHALL occur at the end of the Task.

Table 14 Task Attributes

Sequence ~~flow~~ Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Task ~~can~~ MAY be a target for a Sequence Flow; it can have multiple incoming Flows. ~~An incoming~~ Incoming Flow ~~can~~ MAY be from an alternative path and/or a parallel ~~path. The Flow Condition will determine when the Task will start~~paths.

  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. If the flow needs to be controlled, then the flow should converge on a Gateway that preceeds the Sub-Process (Refer to the section entitled “Gateways” on page 76 for more information on Gateways).

  - If the Task does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Task ~~will~~ MUST be instantiated when the process is instantiated.

- A Task ~~can~~ MAY be a source for a Sequence Flow; it can have multiple outgoing Flows. If there are multiple outgoing Sequence Flows, then this means that a separate parallel path is being created for each Flow.

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 (AND-Split) and the number of Tokens in the group

- 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 ~~will~~ MUST be completed.

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

Note: All Message Flows described here 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 ~~can~~ MAY be the target for Message Flows; it can have zero or one incoming Message Flows.
- A Task ~~can~~ MAY be a source for a Message Flow; it can have zero or more outgoing Message Flows.

Mapping to Execution Languages

~~The following two sections describe how the use of Tasks will map to BPEL4WS and BPML, respectively.~~

~~**BPEL4WS**~~

- ~~A Receive Type Task will be mapped to a BPEL4WS *receive*.~~
  - ~~The Instantiate property will be mapped to the *createInstance* element of the *receive* element. True will be mapped to *yes* and False will be mapped to *no*.~~
- ~~A Send Type Task will be mapped to a BPEL4WS *reply* or a BPEL4WS *invoke* (with only the *inputContainer* specified)~~
- ~~A Service Type Task will be mapped to a BPEL4WS *invoke* (with both the *inputContainer* and *outputContainer* specified)~~
- ~~A User Type Task will not have any Message Flows and will map TBD:~~
- ~~The mapping for the Transaction property is TBD.~~
- ~~If the LoopType is Standard then the Task will be wrapped by a BPEL4WS *while* or *until*.~~
  - ~~A Before EvaluateCondition will map to the BPEL4WS *while*.~~
  - ~~An After EvaluateCondition will map to the BPEL4WS *while*. However, to insure that the Task is performed at least once, the *activity* appropriate for the Task Type will be performed first in a *sequence*, which includes the *while*.~~
  - ~~Any value in Maximum will be appended to the LoopCondition. For example with a LoopCondition of “x < 0” and Maximum of 5 (loops), the final expression would be “(x < 0) and (<TaskName>.Counter <= 5).” A BPEL4WS *assign* will be used to update the Counter property.~~

- ~~If the LoopType is ForEach then the mapping is TBD.~~

Refer to the section entitled “Tasks” on page 151 and the section entitled “Tasks” on page 157 for more information about how the Start Event maps to excution languages.

4.4 Gateways

Gateways are modeling elements that are used to control how Sequence Flow will 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 gate--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 accurate, for output flow a Gateway is actually a collection of “Gates” and the behavior a particular Gateway will determine how many of the Gates will be available for the continuation of flow.

A Gateway is a diamond (see Figure 15), which has been used in many flow chart notations for exclusive branching and is familiar to most modelers. Text associated with the Gateway (e.g., its name) can be placed inside the shape, or above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

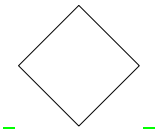


Figure 15 A Gateway

~~BPML~~

- ~~A Receive Type Task will be mapped to a BPML *one-way action*.~~
- ~~A Send Type Task will be mapped to a BPML *notification action*.~~
- ~~A Service Type Task with both an incoming and outgoing Message Flow will be mapped to a BPML *solicit-response action*.~~
- ~~A User Type Task will not have any Message Flows and will map TBD:~~
- ~~The mapping for the Transaction property is TBD.~~
- ~~If the LoopType is Standard then the Task will be wrapped by a BPML *while* or *until*.~~
  - ~~A Before EvaluateCondition will map to the BPML *while*.~~
  - ~~An After EvaluateCondition will map to the BPML *until*.~~
  - ~~Any value in Maximum will be appended to the LoopCondition. For example with a LoopCondition of “x < 0” and Maximum of 5 (loops), the final expression would be “(x < 0) and (<TaskName>.Counter <= 5).” A BPML *assign* will be used to update the Counter property.~~
- ~~If the LoopType is ForEach then the Task will be wrapped by a BPML *foreach*.~~
  - ~~If the Time is Parallel, then the Task will be wrapped in a Nested Process and accessed through BPML *spawn* for each instance and then a BPML *synch* will synchronize them.~~

- ~~Mapping the LoopFlowCondition TBD.~~

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.

Gateways can define all the types of business process Sequence Flow behavior: Decisions/branching (OR-Split; exclusive--XOR, inclusive--OR, and complex), merging (OR-Join), forking (AND-Split), and joining (AND-Join). Thus, while the diamond has been used traditionally for exclusive decisions, BPMN extends the behavior of the diamonds to reflect any type of Sequence Flow control. Each type of Gateway will have an internal indicator or marker to show the type of Gateway that is being used (see Figure 16).

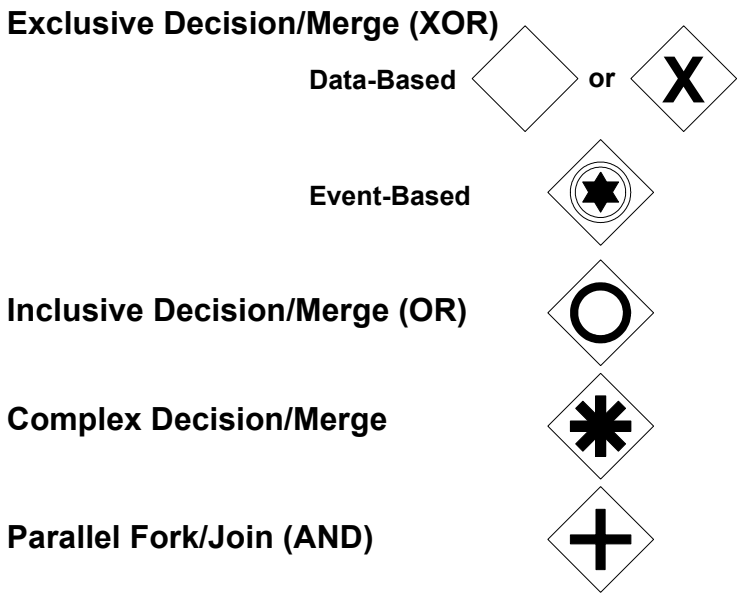


Figure 16 The Different types of Gateways

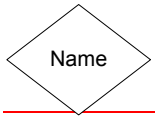
~~**Note**: BPML *request response action* is not mapped to a BPMN Task. Although this type of *action* is an atomic activity in BPML, additional activities can be performed before the *action* is complete. Because addtional work can be done for the response to the request, this will be represented as a Sub-Process in BPMN.~~

~~4.5 Decisions~~

~~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 -- refer to the section entitled "Forking (AND-Split)" on page 86). A Decision is not an activity from the business process perspective, but is an object that controls the 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. Each Decision Alternative is paired with a single outgoing Sequence Flow. When an Alternative 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 Alternative. The Sequence Flows themselves act only as the path through which the Token travels; they do not have their own conditions that can determine whether they are traveled or not.~~

~~A Decision is a diamond (see Figure 15), which has been used in many flow chart notations and is familiar to most modelers. Text~~ The internal indicator associated with the ~~Decision (e.g., its name)~~ Gateway can be placed inside ~~the shape, or above or below~~ the shape, in any ~~direction~~ size or location, depending on the preference of the modeler or modeling tool vendor.



~~Figure 17 A Decision~~

The Gateways will control the flow of both diverging and/or converging Sequence Flow. That is, a particular Gateway could have multiple incoming Sequence Flow and multiple outgoing Sequence Flow at the same time. 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.

4.5.1 Common Gateway Features

Common Gateway Attributes

The following table displays the attributes common for all types of Gateways, and which extends the set of common object elements (see Table 5):

Attributes	Description
GatewayType: (XOR | OR | Complex | AND): XOR	GatewayType is by default XOR. The GatewayType MAY be set to OR, Complex, or AND. The type will determine the behavior of the Gateway, both for incoming and outgoing Sequence Flow, and will determine the internal indicator (as shown in Figure 16).

Table 15 Common Gateway Attributes

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. Refer to the section entitled "Sequence Flow Rules" on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Gateway MAY be a target for a 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 (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 particular Data-Based Exclusive Gateway MAY have both multiple incoming and outgoing Sequence Flow.

Message Flow Connections

This section applies to both Data-Based and Event-Based Exclusive Gateways. Refer to the section entitled "Message Flow Rules" on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

- An Gateway MAY NOT be a target for a Message Flow.
- An Gateway MAY NOT be a source for a Message Flow.

~~**Note**: Although the shape of the Decision 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 Decision shape.~~

4.5.2 Exclusive Gateways (XOR)

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—refer to the section entitled "Forking Flow" on page 93). A Decision is not an activity from the business process perspective, but is one of a set of Gateways that control 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 an 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.

Decisions come in ~~two~~ three basic types: ~~Exclusive~~ Exclusive, Inclusive and ~~Inclusive~~Complex. The following two sections define these two types.

~~4.5.3 Exclusive~~

The Exclusive Decision has two or more outgoing ~~Message~~ 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~~ alternative paths for the Token to take as it traverses the Flows. There are two types of Exclusive Decisions: Data-Based and Event-Based.

Data-Based

The Data-Based Exclusive ~~Decisions~~ Gateways are the most commonly used ~~type~~type of Gateways. We will also refer to them as ~~just~~ Decisions. The set of ~~Alternatives~~ Gates for Data-Based Exclusive Decisions are based on condition expressions. These expressions ~~evaluate~~ use the ~~current~~ values of process data to determine which path should be taken (hence the name Data-Based). The ~~conditions should be evaluated in a specific order. The first one that evaluates as true will determine~~ Data-Based Exclusive Gateway is distinguished from other Gateways by the ~~Sequence Flow that will be taken. One~~ lack of an internal indicator (see Figure 18) or the ~~condition expressions may be~~ use of an indicator that is shaped like an "~~default,~~X" ~~and~~ that is placed within the ~~last condition evaluated~~Gateway diamond (see Figure 19). ~~This means that if none~~ A BDP SHOULD be consistent in the use of the ~~other condition expressions is true at runtime~~"X" internal indicator. That is, ~~then the default expression will be chosen – along~~ a diagram should not have some Gateways with ~~its associated Sequence Flow~~an indicator and some Gateways without an indicator.

~~**Note**: the Default Alternative for a Data Decision Type may be defined as being mandatory in a future version of the specification. This is an open issue. Refer to the section entitled "Open Issues" on page 137 for a complete list of the issues open for BPMN.~~

~~Although the Decision actually contains the condition expressions, they can be displayed on the outgoing Sequence Flows (see Figure 18). While shown on the Sequence Flow, they are not actually a property of the Sequence Flows.~~

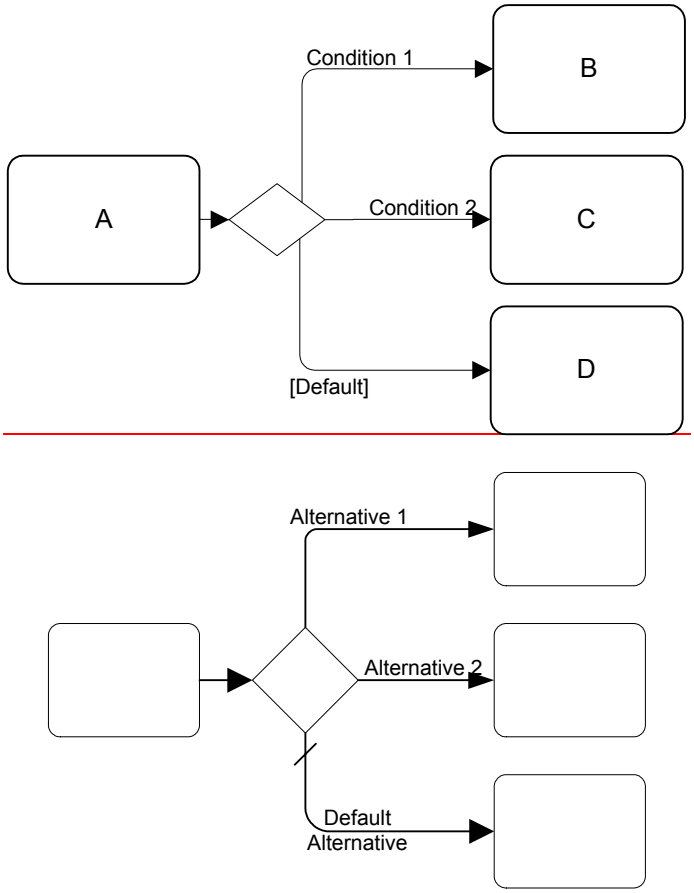


Figure 18 An Exclusve Data-Based Decision (Gateway) Example without the Internal Indicator

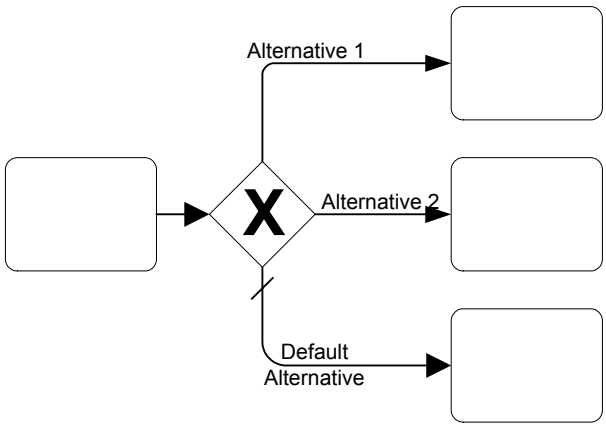


Figure 19 A Data-Based Exclusive Decision (Gateway) Example with the Internal Indicator

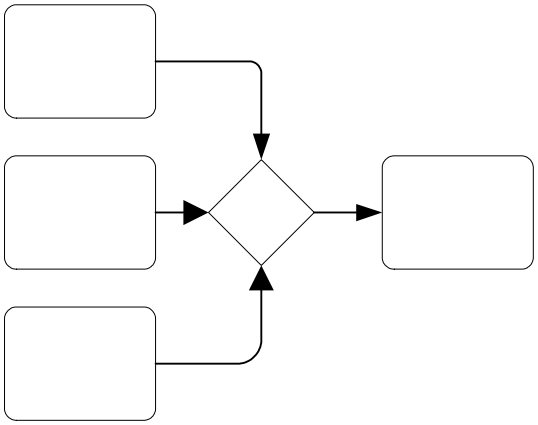


Figure 20 An Exclusve Merge (Gateway) Example without the Internal Indicator

The conditions for the alternative Gates should be evaluated in a specific order. 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,” 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. There are a couple of possibilities: 1) the flow can reach an implicit End Event (this is how a UML solution would work), or 2) the flow can skip the Decision, determine where all alternative paths will merge back together and then continue from that point (this is how a BPEL4WS or BPML solution would work). In either case, this would be a case of implicit flow through the Process. However, BPMN does specify that there can be no 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.

Attributes

The following table displays the attributes for an Data-Based Exclusive Gateway. These attributes only apply if the GatewayType attribute is set to XOR. The following attributes extend the set of common Gateway elements (see Table 15):

Attributes	Description
XORType: (Data | Event): Data	XORType is by default Data. The GatewayType MAY be set to Event. Since Data-Based XOR Gateways is the subject of this section, the attribute MUST be set to Data for the attributes and behavior defined in this section to apply to the Gateway.
Gate *: GateID	There MAY be zero or more Gates. If there are zero or only one incoming Sequence Flow (i.e, the Gateway is acting as a Decision), then there MUST be at least one Gate. In this case, if there is no DefaultGate, then there MUST be at least two Gates.
OutgoingSequenceFlow +: SequenceFlowID	Each Gate MUST have an associated Sequence Flow. The Sequence Flow MUST have a Condition Expression A Gate MAY have multiple Sequence Flow, but the Condition expression for each Sequence Flow MUST be exactly the same. If there is only one Gate (i.e., the Gateway is acting only as a Merge), then then Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for each Gate.
DefaultGate ?: GateID	A Default Gate MAY be specified.
OutgoingSequenceFlow +: SequenceFlowID	If there is a DefaultGate, the it MUST have an associated Sequence Flow. It MAY have multiple. The Sequence Flow SHALL have the Default Indicator (see Figure 18 - this depends on the resolution of Issue 67). The Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for the DefaultGate.

Table 16 ~~A~~ Data-Based ~~Decision Example~~Exclusive Gateway Attributes

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in the section entitled “Common Gateway Sequence Flow Connections” on page 79. Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

To define the exclusive nature of this Gateway’s behavior for converging Sequence Flow:

- If there are multiple incoming Sequence Flows, only one of them will be used to continue the flow of the Process. That is,
  - Process flow SHALL continue when the first signal (a Token) arrives from any of a set of Sequence Flows.

- Signals from other Sequence Flow within that set may arrive, but they SHALL NOT be used to continue the flow of the Process.
  - The Process flow SHALL NOT wait for any of the other signals that may arrive before continuing. <should we specify this or leave it open>

Note: Incoming Sequence Flow that have a source that is a downstream activity (that is, is part of a loop) will be handled differently than those that have an upstream source. They will be considered as part of a different set of Sequence Flow from those Sequence Flow that have a source that is an upstream activity. Each set is handled independently.

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

- If there are multiple outgoing Sequence Flow, then only one Gate (or the DefaultGate) SHALL be selected during performance of the Process.
  - The Gate SHALL be chosen based on the Condition expression 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 Condition is evaluated as "TRUE," then that Gate SHALL be chosen and any Gates remaining on the list SHALL NOT be evaluated.
    - If none of the Conditions 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 Conditions are evaluated as "TRUE," then the Process is considered to have an invalid model.

Mapping to Execution Languages

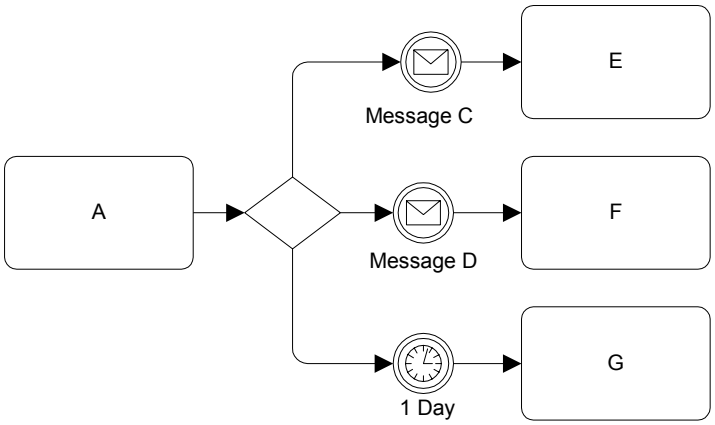
Refer to the section entitled "Data-Based" on page 152 and the section entitled "Data-Based" on page 158 for more information about how the Start Event maps to excution languages.

Event-Based

Event-Based Exclusive ~~Decisions~~ Gateways are a ~~fairly new~~ recent development in Business Process Management and will map to the BPEL4WS *pick* or BPML *choice* elements. On the input side, their behavior is the same as a Data-Based Exclusive Gateway (refer to the section entitled "Data-Based" on page 80 above). ~~The~~ On the output side, the basic idea is that this Decision represents a branching point in the process where the ~~Alternatives~~ alternatives are based on an ~~Intermediate Event~~ events that occurs at that point in the Process. ~~The~~ A specific ~~Intermediate Event~~event, usually the receipt of a ~~message type~~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 completely 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 type Receive or an Intermediate Event of type Message. In addition to Messages, other types of Intermediate Events can be used, such as Timers and ~~Process Errors~~Exceptions.

The Event-Based Exclusive Decisions are configured by using the ~~Decision~~ Gateway shape with an internal marker or indicator that is the same as the Multiple Intermediate Event and having outgoing Sequence Flows target a Task of type Receive or an Intermediate Event (see Figure 22 and Figure 23). All of the outgoing Sequence Flows must ~~target an Intermediate Event~~have this type of target; there cannot be a mixing of condition expressions and Intermediate Events for a given Decision.



~~Figure 21 An Event-Based Decision Example~~

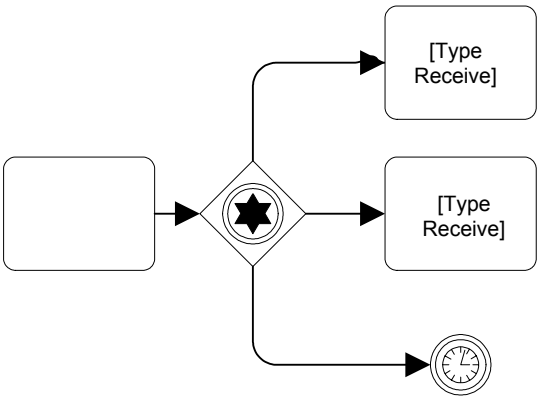


Figure 22 An Event-Based Decision (Gateway) Example Using Receive Tasks

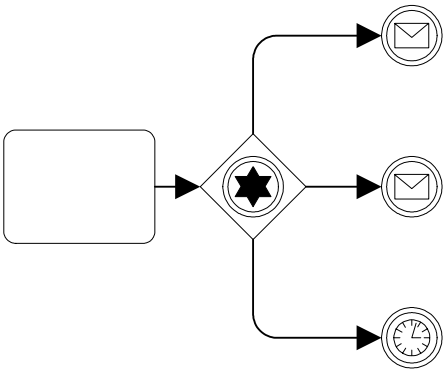


Figure 23 An Event-Based Decision (Gateway) Example Using Message Events

To relate the Event-Based Exclusive Decision to BPEL4WS or BPML, the Decision diamond marks the location of a BPEL4WS *pick* or a BPML *choice* and the Intermediate Events that follow the Decision become the event handlers of the *pick* or *choice*. The activities that follow the Intermediate Events become the contents of the *activity sets* for the event handlers. The boundaries of the activity sets is actually determined by the

configuration of the process; that is, the boundaries extend to where all the alternative paths are finally joined together (which could be the end of the Process).

Attributes

~~The following table displays the identified attributes of an Exclusive Decision:~~

Attributes	Description
Name	Name is a property that is text description of the Decision.
DecisionType: (Data | Event): Data	DecisionType is a property and is by default Data.
(Data) **Alternative** +: Expression	If the type is Data, there must be one or more Alternatives with their expressions.
OutgoingSequenceFlow: SequenceFlowName	Each Alternative must have an associated Sequence Flow.
Assign +: Expression	Zero or more assignments can be made for each Alternative.
(Data) **DefaultAlternative**?	If the type is Data, then a Default Alternative may be specified.
OutgoingSequenceFlow: SequenceFlowName	The Default Alternative must have an associated Sequence Flow.
Assign +: Expression	Zero or more assignments can be made for the DefaultAlternative.
(Event) **Alternative** 2+: OutgoingSequenceFlow	If the type is Event, then two or more Alternatives are defined as Sequence Flows and their targets must be an Intermediate Event. The Intermediate Events must be of type Message, Timer, or Fault. Only one of the Events can be of type Timer, however.
Target: EventName	The targets of the Sequence flow must be an Intermediate Event
IncomingSequenceFlow *: SequenceFlowName	One or more incoming Sequence Flows can be idenitified for the Decision.
FlowCondition: (One | All | Complex) : All	If there is more than one, then a Flow Condition must be set. A Flow Condition of One means that the Decision will be evaluated when one Token arrives on any of the Flows. The process will continue and all other Tokens arriving at the Decision will be consumed. A Flow Condition of All means that a Token must arrive from all incoming Flows before the Decision can be evaluated.
Complex: Expression	A complex Flow Condition can be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process data. The expression will determine when the Decision will be evaluated.
Pool ?: PoolName	If Pools are used, then the PoolName must be added to the Decision to identify its location.
Lane ?: LaneName	If that Pool has more than one Lane, then the LaneName must be added.
Association *	Zero or more Associations can be associated with the Decision.
Documentation ?	The modeler can add optional text documentation about the Decision.

The following table displays the attributes for an Event-Based Exclusive Gateway. These

attributes only apply if the GatewayType attribute is set to XOR. The following attributes extend the set of common Gateway elements (see Table 15):

Attributes	Description
XORType: (Data | Event): Event	XORType is by default Data. The GatewayType MAY be set to Event. Since Event-Based XOR Gateways is the subject of this section, the attribute MUST be set to Event for the attributes and behavior defined in this section to apply to the Gateway.
Gate 2+: GateID	There MUST be two or more Gates. (Note that this type of Gateway does not act *only* as a Merge--it is always a Decision, at least.)
OutgoingSequenceFlow: SequenceFlowID	Each Gate MUST have an associated Sequence Flow. The Sequence Flow MUST NOT have a Condition Expression.
Target: ObjectID	The targets of the Sequence flow MUST be an Intermediate Event or a Task of type Receive. Intermediate Events of type Compensation and Multiple SHALL NOT be allowed as a Target. If a Receive Task is the Target for one Alterative, then a Message Intermediate Event SHALL NOT be allowed for Targets of other Gates.
Assign *: Expression	Zero or more assignments MAY be made for each Gate.

Table 17 Event-Based Exclusive Gateway Attributes

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in the section entitled “Common Gateway Sequence Flow Connections” on page 79. Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

To define the exclusive nature of this Gateway’s behavior for converging Sequence Flow:

- If there are multiple incoming Sequence Flows, only one of them will be used to continue the flow of the Process. That is,
  - Process flow SHALL continue when the first signal (a Token) arrives from any of the set of Sequence Flows.
  - Signals from other Sequence Flow within that set MAY arrive, but they SHALL 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 Flow from the Sequence Flows that have an upstream. Each set of Sequence Flows are treated independently.

- The outgoing Sequence Flow MUST NOT have a Condition Expression.
- The Target of the Gateway's outgoing Sequence Flows MUST be one of the following objects:
  - Task with the Type attribute set to Receive.
  - Intermediate Event with the Trigger attribute set to Message, Timer, Rule, Exception, or Link.
    - If one Gate Target is a Task, then an Intermediate Event of Type Message MAY 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.

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

- Only one Gate SHALL be selected during performance of the Process.
  - The Gate SHALL be chosen based on the Targets 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 SHALL NOT be evaluated (i.e., their Targets will be disabled).

Basically, the first thing that happens will determine the path that the Process will then follow, to the exclusive of the other paths.

<???? Should we mention what happens if another message arrives? Is it ignored? Is it an Error? ????>

Mapping to Execution Languages

Refer to the section entitled "Event-Based" on page 152 and the section entitled "Event-Based" on page 158 for more information about how the Start Event maps to excution languages.

4.5.4 Inclusive Gateways (OR)

The Inclusive Gateway will control the flow of both diverging and converging Sequence Flow.

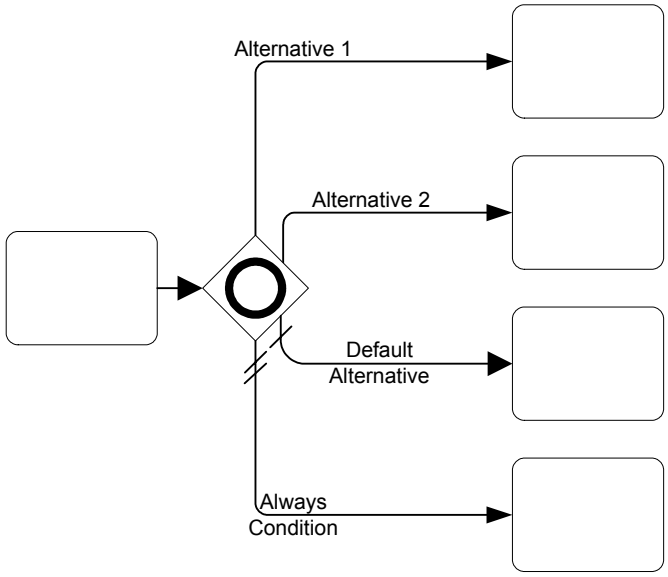


Figure 24 An Inclusive Decision (Gateway) Example

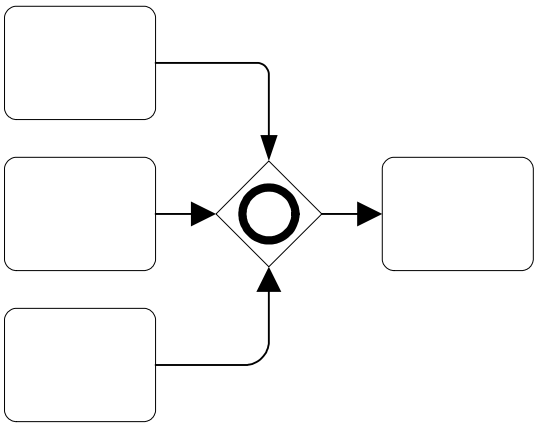


Figure 25 An Inclusive Merge (Gateway) Example

Zero to all of the outgoing Sequence Flows from an Inclusive Decision may be taken during the performance of the Process. Thus, an Inclusive Decision is a hybrid between a Fork (AND-Split) and a Decision (OR-Split). 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. BPMN extends this concept by insisting that there be a default Flow that is followed if none of the other outgoing Flows are taken. This insures that the process behavior is fully covered through the Sequence Flows. Thus, a BPMN Inclusive Decision really will have one or more of the outgoing Flows taken at runtime.

~~Decision~~ *Attributes*

The following table displays the attributes for an Inclusive Gateway. These attributes only apply if the GatewayType attribute is set to OR. The following attributes extend the set of common Gateway elements (see Table 15):

Attributes	Description
Gate *: GateID	There MAY be zero or more Gates. If there are zero or only one incoming Sequence Flow (i.e, the Gateway is acting as a Decision), then there MUST be at least one Gate. In this case, if there is not a DefaultGate or AlwaysGate, then there MUST be at least two Gates.
OutgoingSequenceFlow +: SequenceFlowID	Each Gate MUST have an associated Sequence Flow. The Sequence Flow MUST have a Condition Expression A Gate MAY have multiple Sequence Flow, but the Condition expression for each Sequence Flow MUST be exactly the same. If there is only one Gate (i.e., the Gateway is acting only as a Merge), then then Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for each Gate.
DefaultGate ?: GateID	A Default Gate MAY be specified.
OutgoingSequenceFlow: SequenceFlowName	If there is a DefaultGate, then it MUST have an associated Sequence Flow. It MAY have multiple. The Sequence Flow SHALL have the Default Indicator (see Figure 24 - this depends on the resolution of Issue 67). The Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for the DefaultGate.
AlwaysGate ?: GateID	An Always Gate MAY be specified.
OutgoingSequenceFlow: SequenceFlowName	If there is an AllwaysGate, then it MUST have an associated Sequence Flow. It MAY have multiple. The Sequence Flow SHALL have the Always Indicator (see Figure 24 - this depends on the resolution of Issue 67). The Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for the AlwaysGate.

Table 18 Inclusive Gateway Attributes

Sequence ~~flow~~ Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in the section entitled “Common Gateway Sequence Flow Connections” on page 79. Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- ~~A Data-Based Exclusive Decision can be a target for a Sequence Flow; it can have multiple incoming Flows. An incoming Flow can be from an alternative path or a parallel path.~~
  - ~~If the Decision does not have an incoming Sequence Flow, and there is no Start Event for the Process, then the Decision will be evaluated when the process is instantiated.~~

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

- 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 first signal (a Token) arrives from any of the set of Sequence Flows.
  - Signals from other Sequence Flow within that set MAY arrive, but they SHALL NOT be used to continue the flow of the Process.

- ~~An Exclusive Decision can be a source for a Sequence Flow; it must have two or more outgoing Flows. One of these outgoing Sequence Flows must be a "default" Sequence Flow. The non-default outgoing Flows are evaluated independently to determine if that path will be taken. If none of the non-default Flows taken, then the default Flow will be taken.~~

Note: Incoming Sequence Flow 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 Flow from those Sequence Flow that have a source that is an upstream activity.

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

- 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.
    - If none of the Conditions for the Gates are evaluated as "TRUE," then the DefaultGate SHALL be chosen.
- The AlwaysGate SHALL be chosen regardless of the results of the evaluation of any Gates.

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

Mapping to Execution Languages (not completed yet)

Refer to the section entitled “Inclusive (not completed yet)” on page 153 and the section entitled “Inclusive (not completed yet)” on page 159 for more information about how the Start Event maps to excution languages.

4.5.5 Complex Gateways

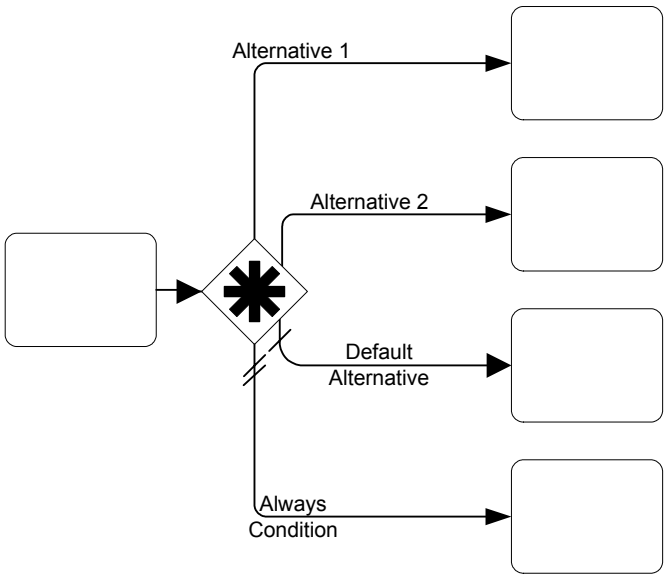


Figure 26 A Complex Decision (Gateway) Example

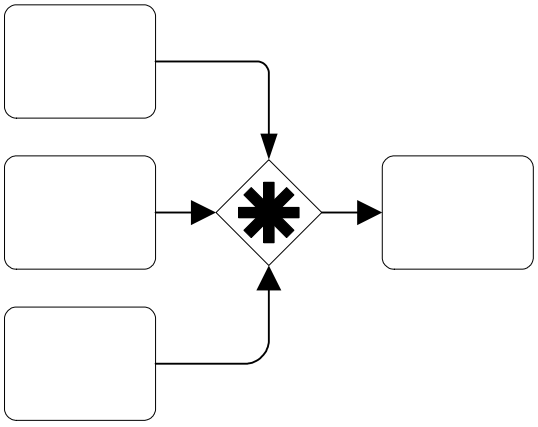


Figure 27 A Complex Merge (Gateway) Example

Attributes

The following table displays the attributes for a Complex Gateway. These attributes only apply if the GatewayType attribute is set to OR. The following attributes extend the set of common Gateway elements (see Table 15):

Attributes	Description
Gate +: GateID	There MAY be zero or more Gates. If there are zero or only one incoming Sequence Flow, then there MUST be at least one Gate. In this case, if there is not a DefaultGate or AlwaysGate, then there MUST be at least two Gates.
OutgoingSequenceFlow +: SequenceFlowID	Each Gate MUST have an associated Sequence Flow. The Sequence Flow MUST have a Condition Expression. A Gate MAY have multiple Sequence Flow, but the Condition expression for each Sequence Flow MUST be exactly the same. If there is only one Gate (i.e., the Gateway is acting only as a Merge), then then Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as “True.”
Assign *: Expression	Zero or more assignments MAY be made for each Gate.
DefaultGate ?	A Default Gate MAY be specified.
OutgoingSequenceFlow: SequenceFlowName	If there is a DefaultGate, then it MUST have an associated Sequence Flow. It MAY have multiple. The Sequence Flow SHALL have the Default Indicator (see Figure 24 - this depends on the resolution of Issue 67). The Sequence Flow SHALL NOT have a Condition expression.
Assign *: Expression	Zero or more assignments MAY be made for the DefaultGate.
AlwaysGate ?	An Always Gate MAY be specified.
OutgoingSequenceFlow: SequenceFlowName	If there is an AlwaysGate, then it MUST have an associated Sequence Flow. It MAY have multiple. The Sequence Flow SHALL have the Always Indicator (see Figure 24 - this depends on the resolution of Issue 67). The Sequence Flow SHALL NOT have a Condition expression.
Assign *: Expression	Zero or more assignments MAY be made for the AlwaysGate.
InletCondition: Expression	
GateCondition: Expression	If there are Multiple incoming Sequence Flow, complex Flow Condition can be set by the modeler. This will consist of an expression that can reference Sequence Flow names and or Process data. The expression will determine when the Task will start.

Table 19 Complex Gateway Attributes

~~Message~~ Sequence Flow Connections

~~Refer to the section entitled "Message Flow Rules" on page 25 for the entire set of objects and how the may be source or targets of Sequence Flows.~~

- ~~An Exclusive Decision cannot be a target for a Message Flow.~~
- ~~An Exclusive Decision cannot be a source for a Message Flow.~~

This section extends the basic Gateway Sequence Flow connection rules as defined in the section entitled "Common Gateway Sequence Flow Connections" on page 79. Refer to the section entitled "Sequence Flow Rules" on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

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

- 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 first signal (a Token) arrives from any of the set of Sequence Flows.
  - Signals from other Sequence Flow within that set MAY arrive, but they SHALL NOT be used to continue the flow of the Process.

Note: Incoming Sequence Flow 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 Flow from those Sequence Flow that have a source that is an upstream activity.

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

- 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.
    - If none of the Conditions for the Gates are evaluated as "TRUE," then the DefaultGate SHALL be chosen.
- The AlwaysGate SHALL be chosen regardless of the results of the evaluation of any Gates.

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

Mapping to Execution Languages (not completed yet)

Refer to the section entitled “Complex (not completed yet)” on page 153 and the section entitled “Complex (not completed yet)” on page 159 for more information about how the Start Event maps to excution languages.

4.5.6 Parallel Gateways (AND)

Parallel Gateways provide a mechanism to synchronize parallel flow and to create parallel flow. These Gateways are not required to create parallel flow, 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

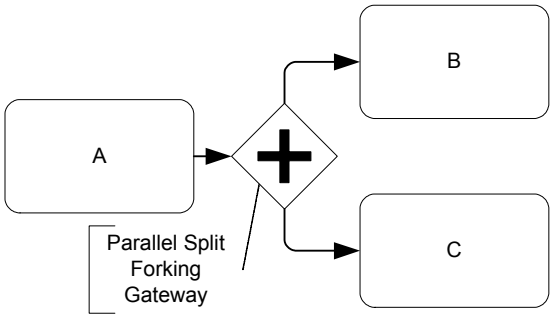


Figure 28 A Parallel Gateway

Parallel Gateways are required for synchronizing parallel flow. Synchronization

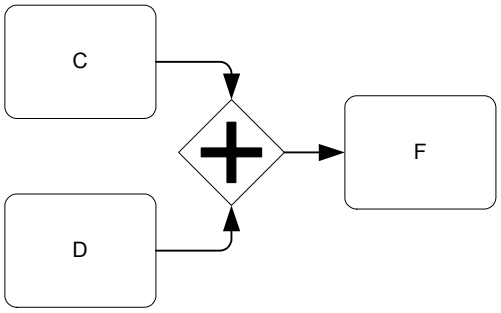


Figure 29 Joining – the joining of parallel paths

Attributes

The following table displays the attributes for a Parallel Gateway. These attributes only apply if the GatewayType attribute is set to AND. The following attributes extend the set of common Gateway elements (see Table 15):

Attributes	Description
Gate *: GateID	There MAY be zero or more Gates. If there are zero or only one incoming Sequence Flow (i.e, the Gateway is acting as a fork), then there MUST be at least two Gates.
OutgoingSequenceFlow: SequenceFlowID	Each Gate MUST have an associated Sequence Flow. The Sequence Flow SHALL NOT have a Condition expression. The Gate Condition will automatically be evaluated as "True."
Assign *: Expression	Zero or more assignments MAY be made for each Gate.

Table 20 Parallel Gateway Attributes

Sequence Flow Connections

This section extends the basic Gateway Sequence Flow connection rules as defined in the section entitled "Common Gateway Sequence Flow Connections" on page 79. Refer to the section entitled "Sequence Flow Rules" on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

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

- If there are multiple incoming Sequence Flows, all of them will be used to continue the flow of the Process. 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).

~~Mapping to Execution Languages~~

~~The following two sections describe how the use of Decisions will map to BPEL4WS and BPML, respectively.~~

~~BPEL4WS~~

- ~~A Data Based Exclusive Decision will map to a BPEL4WS *switch*. Each ConditionExpression will map to the condition for a *switch case*. The Default condition will map to the Switch *otherwise case*.~~
  - ~~The activities that follow the conditions will be included within the *activity* (usually a *sequence*) for that condition. The exact content of the *activity* will depend on the configuration of the Process. Details of how the configuration will be mapped to the activity set can be found in the section entitled "Mapping to Execution Languages" on page 133.~~

- ~~An Event Based Exclusive Decision will map to a BPEL4WS *pick*. Each of the target Intermediate Events will map to the message handlers within the *pick*.~~
  - ~~The activities that follow the Intermediate Events will be included within the *activity* (usually a *sequence*) for that message handler. The exact content of the *activity* will depend on the configuration of the Process. Details of how the configuration will be mapped to the activity set can be found in the section entitled "Mapping to Execution Languages" on page 133. If the Intermediate Event is of type Message, then the first activity of the *activity* will be a *receive*.~~

Note: Incoming Sequence Flow 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 Flow from those Sequence Flow that have a source that is an upstream activity.

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

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

~~**BPML**~~

- ~~A Data Based Exclusive Decision will map to a BPML *switch*. Each ConditionExpression will map to the condition for a *switch case*. The Default condition will map to the Switch *default case*.~~
  - ~~The activities that follow the conditions will be included within the *activity set* for that condition. The exact content of the *activity set* will depend on the configuration of the Process. Details of how the configuration will be mapped to the activity set can be found in the section entitled "Mapping to Execution Languages" on page 133.~~
- ~~An Event Based Exclusive Decision will map to a BPML *choice*. Each of the target Intermediate Events will map to the event handlers within the *choice*.~~
  - ~~The activities that follow the Intermediate Events will be included within the *activity set* for that event handler. The exact content of the *activity set* will depend on the configuration of the Process. Details of how the configuration will be mapped to the activity set can be found in the section entitled "Mapping to Execution Languages" on page 133. If the Intermediate Event is of type Message, then the first activity of the *activity set* will be a *one way action*.~~

Mapping to Execution Languages (not completed yet)

Refer to the section entitled "Parallel (not completed yet)" on page 154 and the section entitled "Parallel (not completed yet)" on page 160 for more information about how the Start Event maps to excution languages.

~~4.5.7 Inclusive~~

~~Zero to all of the outgoing Sequence Flows from an Exclusive Decision may be taken during the performance of the Process. Thus, an Inclusive Decision is a hybrid between a Fork (AND-Split) and a Decision (OR-Split). 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. BPMN extends this concept by insisting that there be a default Flow that is followed if none of the other outgoing Flows are taken. This insures that the process behavior is fully covered through the Sequence Flows. Thus, a BPMN Inclusive Decision really will have one or more of the outgoing Flows taken at runtime.~~

~~**Editor's Note**: the details of the how Inclusive Decisions look and behave is an open issue and will be included in a later version of the specification. Since Inclusive Decisions are a hybrid of forking and splitting, the definition of their behavior may be moved to another section in this specification, depending on how the notation of these Decisions are finalized. Refer to the section entitled "Open Issues" on page 137 for a complete list of the issues open for BPMN.~~

4.6 Pools and Lanes

BPMN has a larger scope than BPEL4WS or BPML, and this scope is expressed in different dimensions. The dimension discussed here has to with defining business processes in a collaborative B2B environment. BPMN uses the concept known as "swimlanes" to help partition ~~and~~ and/organize activities.

BPEL4WS and BPML are focused on a specific private process that is internal to a given Participant (i.e., a company or organization). BPEL4WS also can define an abstract process, but from the point of view of a single participant. 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 different Participants. 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."

The ~~section entitled "Uses of BPMN" on page 15~~section entitled "Uses of BPMN" on page 18 describes the uses of BPMN for modeling private processes and the interactions of processes in B2B scenarios. Pools and Lanes are designed to support these uses of BPMN.

4.6.1 Pool

A Pool is a "swimlane" and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations. It is a square-cornered rectangle that is drawn with a solid single line (as seen in Figure 30). 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~~ 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. Text associated with the Pool (e.g., its name) can be placed inside the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

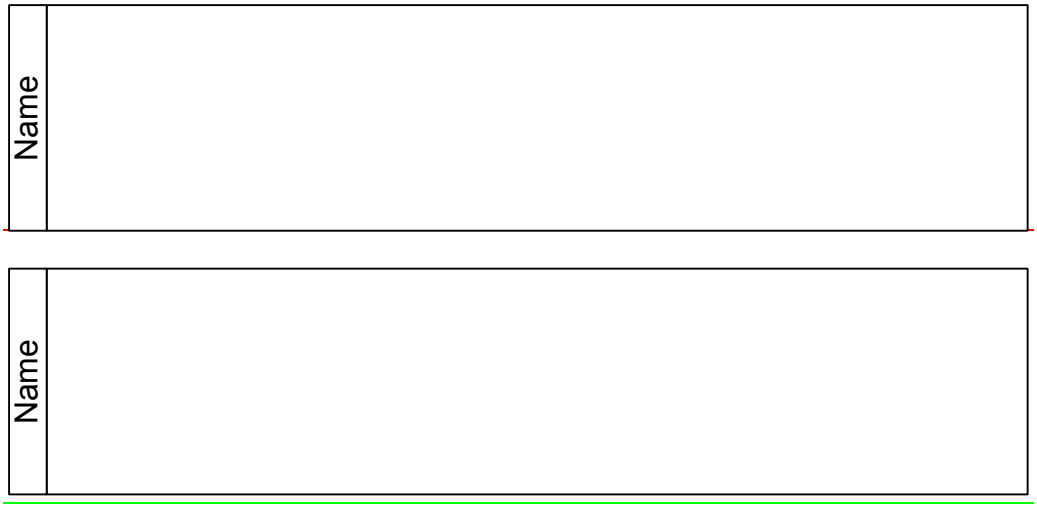


Figure 30 A Pool

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, e.g., in a B2B context, is shown through Message Flows.

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 Flows can attach to its boundaries (see Figure 31).

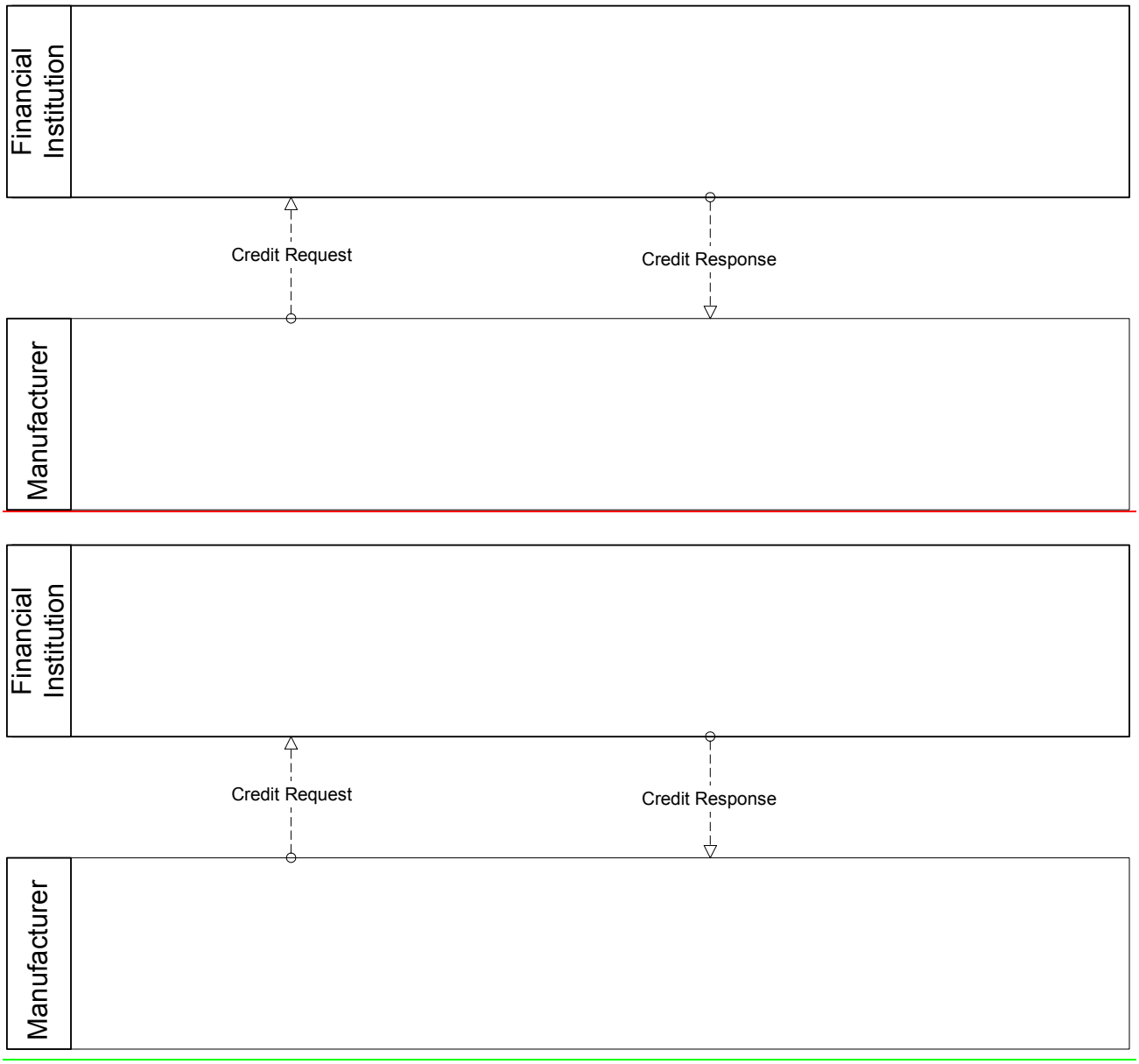


Figure 31 Message Flow connecting to the boundaries of two Pools

For a “White Box” Pool, the activities within are organized ~~within~~ by Sequence Flows. Message Flows can cross the Pool boundary to attach to the appropriate activity (see Figure 32).

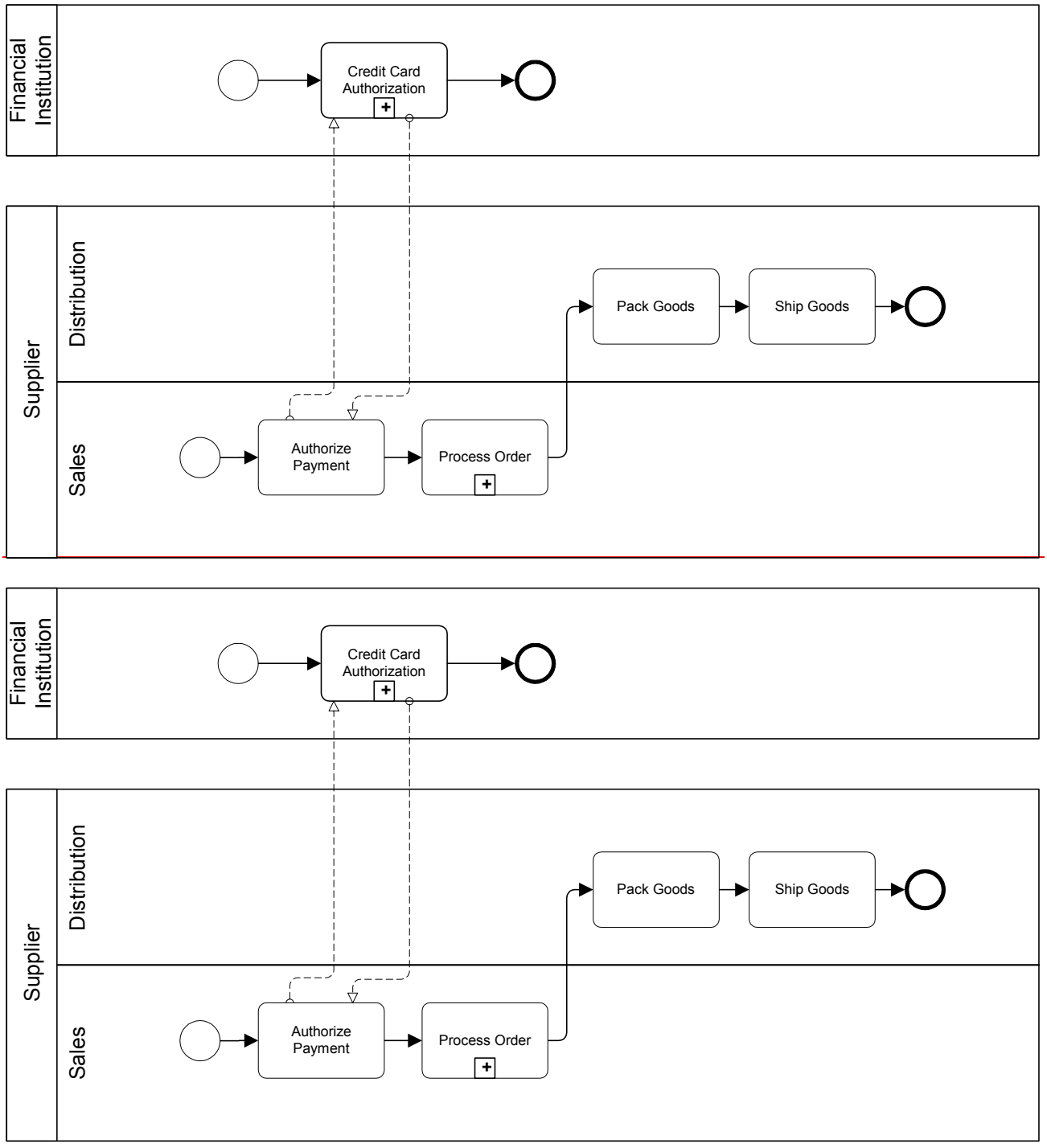


Figure 32 Message Flow connecting to flow objects within two Pools

All BPDs contain at least one Pool. 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, many BPDs may show the “main” Pool without boundaries. That is, 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 may not be surrounded by

the boundaries of a Pool, while the other Pools in the diagram will have their boundary. (see Figure 33)

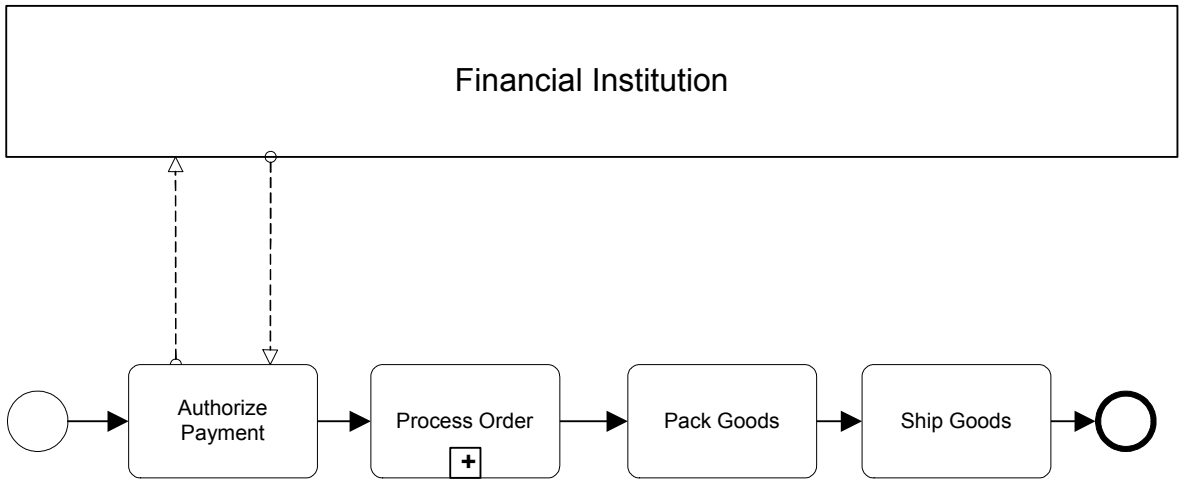


Figure 33 Main (Internal) Pool without boundaries

Attributes

The following table displays the identified attributes of a Pool:

Attribute	Description
Name	Name is a property that is text description of the Pool.
PoolType (Private | Interface | Collaboration): Private	Pool Type is a property that provides information about to which lower-level language the Pool will be mapped. The default type is Private which will be mapped to BPEL4WS or BPML. An Interface Pool is also called the public interface of a process (or other web services) and will be mapped to languages such as WSCI. A Collaboration Pool will have two Lanes that represent business roles (e.g., buyer or seller) and will show the interactions between these roles. These pools will be mapped to languages such as ebXML.
Owner ?	Owner is an optional property that will help identify the point-of-view of the diagram. If the Pool Type is Collaboration, then there is no specific Owner.
IncomingMessageFlow *: MessageFlowName	There can be zero or more incoming Message Flows.
Target (Boundary | FlowObjectName): Boundary	Each Outgoing Message Flow has to have a target specified, which can be either the boundary of the Pool or a flow object within the Pool.
OutgoingMessageFlow *: MessageFlowName	There can be zero or more outgoing Message Flows.
Source (Boundary | FlowObjectName): Boundary	Each incoming Message Flow has to have a source specified, which can be either the boundary of the Pool or a flow object within the Pool.
Lane +: LaneName	There can be one or more Lanes within a Pool. If there is only one Lane, then that Lane shares the name of the Pool and only the Pool name is displayed. If there is more than one, then each Lane has to have its own name and all names are displayed.
Association *	Zero or more Associations can be associated with the Pool.
Documentation ?	The modeler can add optional text documentation about the Pool.

Attribute	Description
PoolType (Private | Interface | Collaboration): Private	Pool Type is an attribute that provides information about which lower-level language the Pool will be mapped. The default type is Private which MAY be mapped to BPEL4WS or BPML. An Interface Pool is also called the public interface of a process (or other web services) and MAY be mapped to languages such as WSCI. A Collaboration Pool 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.

Attribute	Description
Owner ?: String	Owner is an optional attribute that will help identify the point-of-view of the diagram. If the Pool Type is Collaboration, then there is no specific Owner.
Lane *: LaneName	There can be one or more Lanes within a Pool. If there is only one Lane, then that Lane shares the name of the Pool and only the Pool name is displayed. If there is more than one Lane, then each Lane has to have its own name and all names are displayed.

Table 21 Pool Attributes

Mapping to Execution Languages

Pools do not have any specific Mapping to Execution Languages. However, a Pool is associated with a mapping to a specific lower level language. For example, one Pool may encompass a BPEL4WS or BPML document while another Pool might encompass an ebXML BPSS document.

Note: the current specification does not contain mappings to any languages except to the execution definitions in BPEL4WS and to BPML. The mapping to the abstract definitions in BPEL4WS will be included in a later version of the specification.

Note: Interface processes may be allowed to be a Lane within a Pool in a later version of the specification. This is an open issue. Refer to the section entitled “Open Issues” on page 165 for a complete list of the issues open for BPMN.

4.6.2 Lane

A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally (see Figure 34). Text associated with the Lane (e.g., its name) 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.

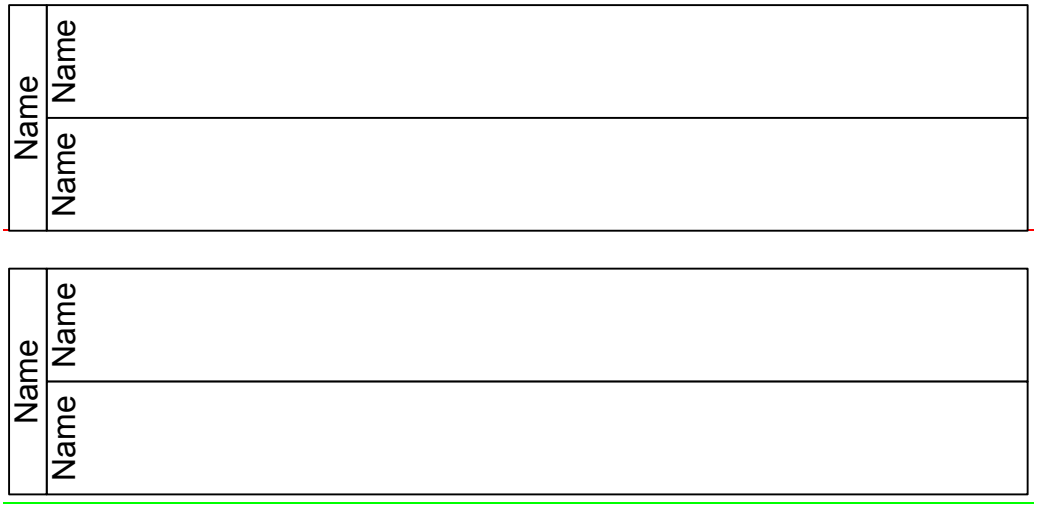


Figure 34 Two Lanes in a Pool

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~~nested 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.

Attributes

The following table displays the identified attributes of a Lane:

Attribute	Description
Name	Name is a property that is text description of the Lane. If the Lane is the only one in the Pool, it will share the name of the Pool.
ParentPool: PoolName	The Parent Pool must be specified. There can be only one Parent.
Association *	Zero or more Associations can be associated with the Lane.
Documentation ?	The modeler can add optional text documentation about the Lane.

Attribute	Description
ID: String	This is a unique ID that identifies the Lane from other objects within the diagram.
Name: String	Name is an attribute that is text description of the Lane. If the Lane is the only one in the Pool, it will share the name of the Pool.
ParentPool: PoolName	The Parent Pool MUST be specified. There can be only one Parent.
ParentLane ?: LaneName	ParentLane is an optional attribute that is used if the Lane is nested within another Lane. Nesting can be multi-level, but only the immediate parent is specified.
Documentation ?: String	The modeler can add optional text documentation about the Lane.

Table 22 Lane Attributes

Mapping to Execution Languages

Lanes do not have any specific Mapping to Execution Languages. They are designed to help organize and communicate how activities are grouped in a business process.

4.7 Data Object

In BPMN, a Data Objects are considered artifacts and not a flow object. They are considered an artifact because they do not have any direct affect on the Sequence Flow or Message Flow of the Process, but they do 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, they can be used to represent many different types of objects, both electronic and physical.

In general, BPMN will not standardize many modeling artifacts. These will mainly be up to modelers and modeling tool vendors to create for their own purposes. However, equivalents of the BPMN Data Object are used by Document Management oriented workflow systems and many other process modeling methodologies. Thus, this object is used enough that it is important to standardize its shape and behavior.

The Data Object is a portrait-oriented rectangle that has its upper-right corner folded over (see Figure 35). Text associated with the Data Object (e.g., its name and/or state) can be placed above or below the shape, in any direction or location, depending on the preference of the modeler or modeling tool vendor.

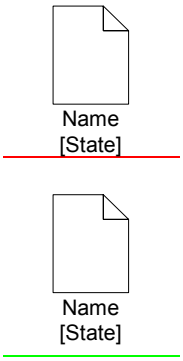


Figure 35 A Data Object

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 Process to another, via a Sequence Flow (see Figure 36). Data Objects will also be associated with Message Flows. They are not to be confused with the message itself, but could be though of as the “payload” or content of some messages.

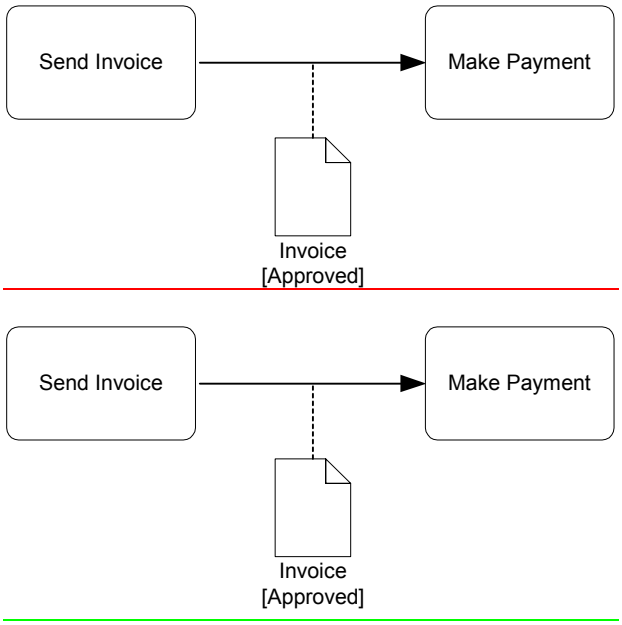


Figure 36 A Data Object associated with a Sequence Flow

In other cases, the same Data Object will be shown as being an input, then an output of a Process (see Figure 37). Directionality added to the Association will show whether the Data

Object is an input or an output. Also, the state ~~property~~ attribute of the Data Object can change to show the impact of the Process on the Data Object.

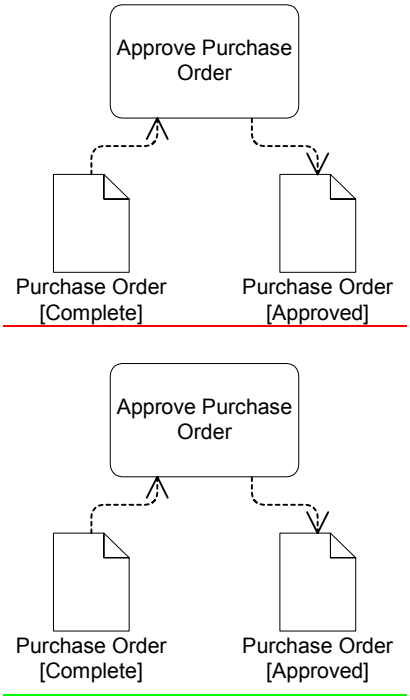


Figure 37 Data Objects shown as inputs and outputs

Attributes

~~The following table displays the identified attributes of a Text Annotation:~~

Attribute	Description
Name	Name is a property that is a text description of the Data Object. Multiple Data Objects can share the same name within one Process.
State ?	State is an optional property that indicates the impact the Process has had on the Data Object. Multiple Data Objects with the same name can share the same state within one Process.
Target: ObjectName	Target is an option property that identifies the object that the Data Object is connected to through an Association.
Documentation ?	The modeler can add optional text documentation about the Data Object.

The following table displays the identified attributes of a Data Object (Note that this is the complete set and it does not extend the set of common object attributes):

Attribute	Description
Id: String	This is a unique ID that identifies the object from other objects within the diagram.
Name: String	Name is an attribute that is text description of the 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.
Pool ?: PoolName	If Pools are used, then the PoolName MUST be added to the object to identify its location.
Lane *: LaneName	If the Pool has more than one Lane, then a LaneName MUST be added. There MAY be multiple Lanes listed if the Lanes are organized in matrix or overlap in a non-nested manner.
Documentation ?: String	The modeler MAY add optional text documentation about the object.

Table 23 Data Object Attributes

Sequence Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Data Object cannot be a target for a Sequence Flow.
- A Data Object cannot be a source for a Sequence Flow.

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Data Object cannot be a target for a Message Flow.
- A Data Object cannot be a source for a Message Flow.

Mapping to Execution Languages

Data Objects do not have any Mapping to Execution Languages. They provide detailed information about how data will interact with the flow objects and flows of Processes.

4.8 Text Annotation

Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN diagram. The Text Annotation object is an open rectangle and can be connected to a specific object on the diagram with an Association (see Figure 38). Text associated with the Annotation can be placed within the bounds of the open rectangle.

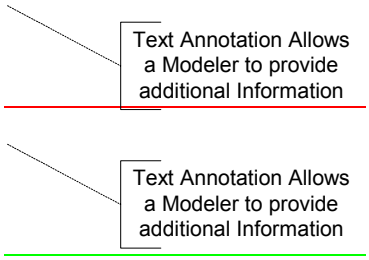


Figure 38 A Text Annotation

Text Annotations do not affect the flow of the Process and do not map to any BPEL4WS or BPML elements.

Attributes

~~The following table displays the identified attributes of a Text Annotation:~~

Attribute	Description
Text: String	Text is a property that is text that the modeler wishes to communicate to the reader of the diagram.
Target ?: ObjectName	Target is an optional property that identifies what object the Annotation is connected to through an Association.

The following table displays the identified attributes of a Text Annotation (Note that this is the complete set and it does not extend the set of common object attributes):

Attribute	Description
Id: String	This is a unique ID that identifies the Text Annotation from other objects within the diagram.
Text: String	Text is an attribute that is text that the modeler wishes to communicate to the reader of the diagram.

Table 24 Text Annotation Attributes

Sequence Flow Connections

Refer to the section entitled “Sequence Flow Rules” on page 31 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Text Annotation cannot be a target for a Sequence Flow.
- A Text Annotation cannot be a source for a Sequence Flow.

Message Flow Connections

Refer to the section entitled “Message Flow Rules” on page 32 for the entire set of objects and how the may be source or targets of Sequence Flows.

- A Text Annotation cannot be a target for a Message Flow.
- A Text Annotation cannot be a source for a Message Flow.

Mapping to Execution Languages

Text Annotations can map to the *documentation* element of BPM execution languages. If the Annotation is associated with a flow object and that object has a straight-forward mapping to a BPM execution language element, then the text of the Annotation will be placed in the *documentation* element of that object. If there is no straight-forward mapping to a BPM execution language element, then the text of the Annotation will be appended to the *documentation* element of the *process*.