



## BPMN and the Business Process Expert, Part 2: Mastering the Notation

**Summary:** A brief summary of the BPMN notation. BPMN describes process orchestration in terms of activities (tasks and subprocesses) connected by sequence flows. Branches, splits, and joins in the flow are modeled by various gateway types. Events specify how processes respond to signals received from external entities or other parts of the same process. Other parts of the notation are loosely specified and used to add business context only. Second of six parts.

**Author:** Bruce Silver

**Company:** Bruce Silver Associates

**Created on:** 5 November 2007

### Author Bio



Dr Bruce Silver is an independent industry analyst and consultant focused on business process management software. He provides training on process modeling with BPMN through [BPMessentials.com](http://BPMessentials.com), the [BPM Institute](http://BPM Institute), and Gartner conferences, and is the author of The BPMS Report series of product evaluations available from the BPM Institute.

In the first installment of this series, we saw why BPMN is important to the Business Process Expert. In this part, we'll look at the notation itself.

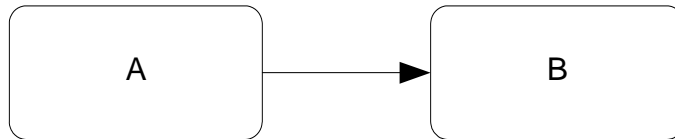
In BPMN there are only three first-class diagram elements, or flow objects:

- *Activity*, a rounded rectangle, representing work performed in the process
- *Gateway*, a diamond, representing flow control logic, such as branching, splits, and joins
- *Event*, a circle, representing a signal that something has happened, either outside the process or inside.

Each of these shapes has various subtypes indicated by an icon or symbol inside, or occasionally by border style. The color of a diagram element has no significance in BPMN.

### Sequence Flows

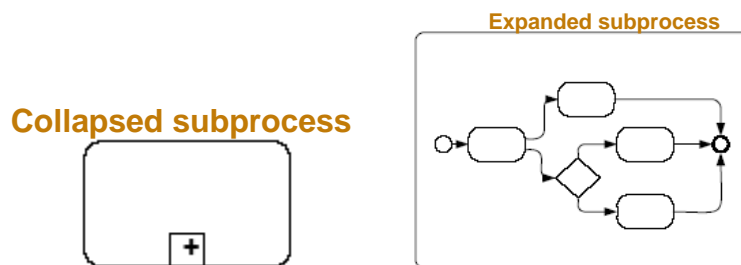
These elements are connected in a process flow by a solid arrow called a *sequence flow*. A sequence flow from activity A to activity B does not signify some user-specified relationship between A and B. It signifies only one thing: After activity A completes, activity B starts, or is enabled to start. A sequence flow variant, called *conditional sequence flow*, and drawn with a mini-diamond on the tail, indicates that the transition from A to B is enabled only if a specified condition is met. But the more common variant, with no diamond on the tail, means the transition is immediate and unconditional.



**Figure 1. Sequence flow means when A ends, B starts.**

## Activities

Steps in the process are indicated by *activities*. BPMN defines two types of activities, task and subprocess. A *task* is an activity that has no component subpart structure of interest to the process model; it is *atomic* in that respect. A *subprocess* is an activity that has component structure of interest. BPMN allows the subprocess to be rendered either *collapsed*, as an opaque rounded rectangle with a + symbol inside, or *expanded*, as an enlarged rounded rectangle inside of which the component structure is drawn in the form of another BPMN diagram. The expansion of a collapsed subprocess may also be drawn on a separate page, which is considered part of the same business process diagram.



**Figure 2. Subprocesses can be rendered either collapsed or expanded.**

Unlike BPEL and similar block languages, BPMN allows sequence flows to loop back to the same activity or a previous activity, as long as it does not cross the boundary of a process or subprocess.

BPMN defines several *task type* attributes. Most modeling tools distinguish supported task types in the diagram by an icon inside the task, an extension allowed by the BPMN spec. The two task types of greatest importance are *user*, meaning a human task, and *service*, meaning an automated task. A *send* task means the process immediately sends a *message*, a signal to an external process or system. A *receive* task means the process pauses and waits to receive a message. Send and receive tasks are thus the same as message events.

An activity, whether task or subprocess, can be specified as repeating. A *loop* task, indicated with a circular arrow icon inside, is like a *While* construct in programming. After the task is performed, a logic condition determines whether to perform it again or to continue. A *multi-instance* (MI) task, indicated with a parallel bar icon inside, is like a *ForEach* construct in programming. N instances of the activity are performed, either sequentially or in parallel, with N generally known in advance.

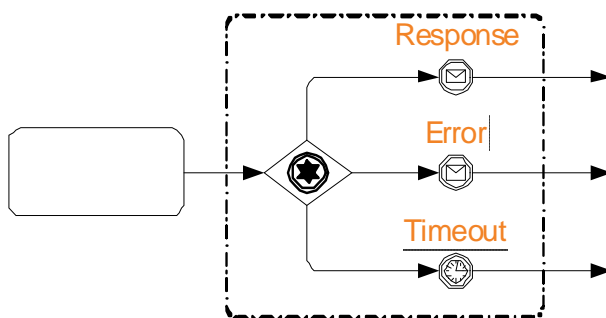
## Gateways

*Gateways* control process flow. A *decision* gateway controls the flow from one incoming sequence flow to one or more outgoing sequence flows. A *merge* or *join* gateway controls the flow from multiple incoming sequence flows to one outgoing sequence flow. Conditional branching, parallel splits, and synchronizing joins all use the same diamond shape. An icon inside the diamond indicates the specific semantics of the gateway. Some

defined gateways are actually redundant, in that a diagram drawn without the gateway has the same process semantics.

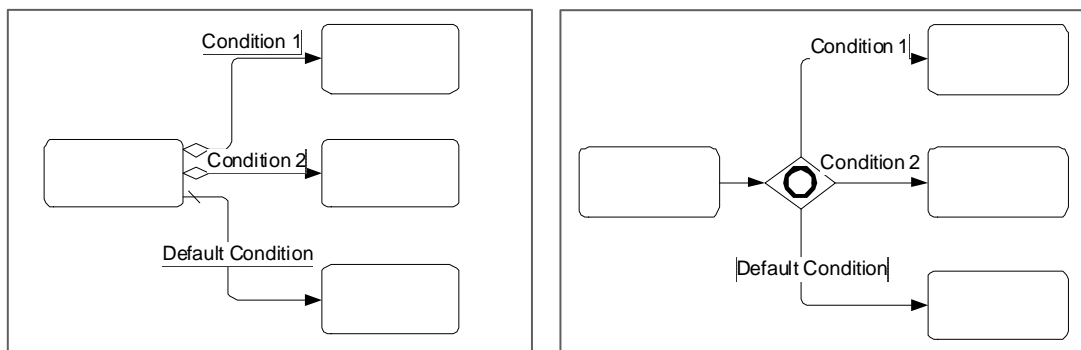
The most common gateway is the *exclusive data-based gateway*, also called an XOR gateway and drawn either with an X inside or no symbol inside. An exclusive decision gateway means that a single outgoing sequence flow is selected by a conditional expression of process data. An exclusive merge gateway is actually redundant, since if the sequence flows into the gateway represent exclusive alternative paths, the uncontrolled flow (without the gateway) means the same thing, and if the sequence flows in are not exclusive alternatives, the exclusive gateway is illegal.

The *exclusive event-based gateway*, more commonly called just *event gateway*, is drawn with a multiple event symbol inside and an intermediate event symbol beginning each outgoing sequence flow. It means a single outgoing sequence flow is selected by the first event to occur, like the *Pick* construct in BPEL. The most common use of event gateways is to listen for a message, with a timeout in case the message does not arrive.



**Figure 3.** Event gateway means take path following the first event to occur.

An *inclusive decision gateway*, some times called OR-gateway and drawn with an O symbol inside, means that *one or more* of the outgoing sequence flows may be enabled, as their enabling conditions are *independent*. For example, enable path 1 if the orderAmount is over \$1000, and enable path 2 if the shipTo is a foreign address. Since one path out of a gateway always must be enabled for any instance, a *default* path, indicated by a tickmark on the sequence flow, is often provided. A *conditional sequence flow*, described earlier, is an alternative representation of this. Conditional sequence flows can only emerge from an activity. Drawing them out of gateways is a commonly seen error.

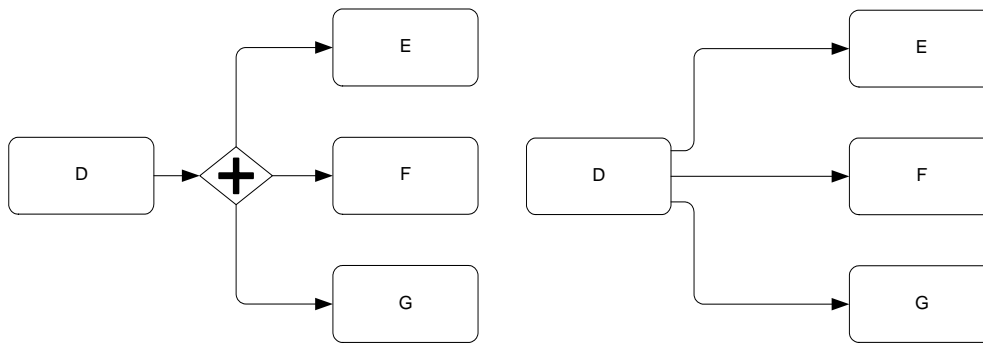


**Figure 4.** Conditional sequence flow (left) and Inclusive gateway (right) are equivalent, used to represent independent enabling conditions.

Merging paths that may or may not be exclusive alternatives, such as those coming out of an inclusive gateway, must use an *inclusive merge gateway*, or OR-join. All paths into the

merge that are enabled must arrive at the gateway before the flow continues. An exclusive merge or parallel merge would be incorrect.

A *parallel split* gateway, also called an AND gateway and drawn with a + symbol inside, means that *all* of the outgoing paths are enabled unconditionally. It is actually redundant, since drawing the outgoing sequence flows without the gateway means the same thing. A *parallel join gateway*, also called AND-join or synchronizing join, means all of the incoming sequence flows must arrive at the gateway before the flow continues. Unlike the parallel split, it is not redundant, but required to merge parallel flows.



**Figure 5. Parallel split gateway (left) is redundant, since uncontrolled flow (right) means the same thing.**

A *complex gateway*, with a \* symbol inside, indicates flow control behavior beyond those achievable with the other gateway types. A use case for complex decision is where the set of multiple outgoing sequence flows is determined by a logical condition. It has somewhat more utility as a merge gateway. One use case is the *voting pattern*, where N out of M paths must arrive at the gateway (perhaps with a particular data value, such as a Yes vote) in order for the flow to continue. The second is the *discriminator pattern*, where the first path to arrive at the gateway is passed through and the others discarded.

In all of these gateway types, the logic conditions describing the semantics may be specified in a BPMN *attribute* of the gateway. In fact, the BPMN spec says the attribute **MUST** be provided, but this requirement is often ignored in actual tools, with good reason. Simple labels applied to the sequence flows are more user-meaningful in the diagram, and BPMN tools that provide executable artifacts typically provide their own dialogs for defining the execution semantics, outside of BPMN.

## Events

Events, circles indicating something has happened, are what makes BPMN different from conventional process modeling notations. BPMN distinguishes start, intermediate, and end events by their border style. *Start events*, drawn with a thin border, indicate the start of a process or subprocess. Multiple start events are allowed in BPMN, but the semantics are ambiguous and the spec discourages use of that pattern.

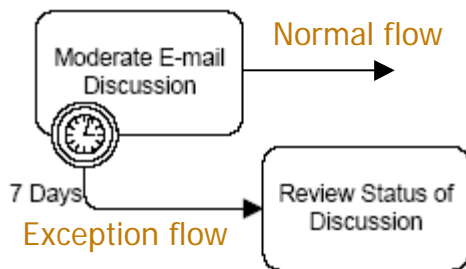
*End events*, drawn with a thick border, indicate the end of a path in a process or subprocess. All enabled paths of a process or subprocess must reach an end event for the process or subprocess to complete normally. In other words, there is an implicit OR-join of all end events. Multiple end events in a process or subprocess are commonly seen. They simplify the drawing, differentiation of successful and failed end states, and allow some end states to throw events and others not to do so.



**Figure 6. Message event in sequence flow means either send the message or wait for the message. In BPMN 1.1, throwing and catching icons are different.**

*Intermediate events*, drawn with a double border, occur after a process starts but before it ends. Their semantics depend on their placement in the diagram. When drawn *in sequence flow*, i.e., with sequence flow both into and out of the event, an intermediate event can mean either *throw* the event signal or *wait to catch* an event signal, depending on the icon inside.

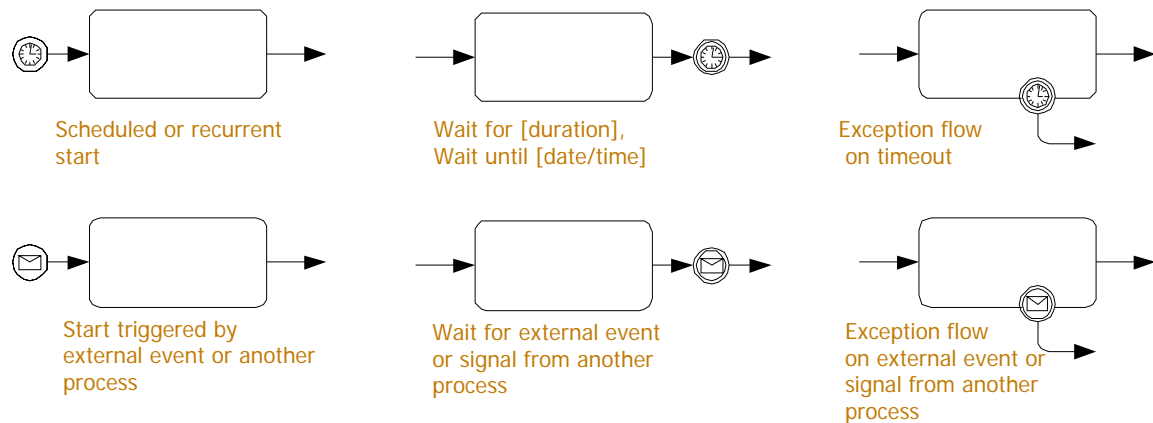
When drawn *attached* to the boundary of an activity, it means if the event occurs, *interrupt* the activity and continue on the sequence flow out of the event, called the *exception flow*. Attached events are only active while the activity they are attached to are active. Events arriving earlier or later are ignored. Thus if an activity with an attached event completes before the event occurs, the sequence flow directly out of the activity, called the *normal flow*, is taken. If the attached event occurs, the *exception flow* is taken.



**Figure 7. Attached event means if the event occurs, interrupt the activity and proceed on exception flow. If activity completes before the event occurs, take the normal flow.**

The specific signal, or event type, is indicated by an icon inside the circle. BPMN defines many types, but only a small subset is commonly used. A *timer event* is indicated with a clock icon. Timer start means start the process on a prescribed schedule. Timer intermediate in sequence flow means wait for a specified duration, or wait until a specified date/time. Attached timer event means interrupt the activity if it has not completed by a specified duration (after the activity starts) or a specified date/time.

*Message event* is indicated with an envelope symbol. In BPMN a message event means any signal to or from *outside* the process. Message start means the process is triggered by receipt of a message. Message end means the process sends a message signal when the end event is reached. Message intermediate event in sequence flow can mean either the process sends the message signal or pauses and waits for the message signal, exactly the same as send and receive tasks. The ambiguity of the symbol is removed in BPMN 1.1, which shows “throwing” event icons as black with white lines and “catching” events as white with black lines. A message signal can only be thrown to a target *outside* the process; it may not be caught by a message event within the same BPMN process.



**Figure 8. Most commonly used timer and message event patterns.**

Attached message event means if the message signal occurs while the activity it is attached to is active, interrupt the activity and proceed on the exception flow. In this way, the activity explicitly defines the context for the event. Process modelers may even wrap a fragment of the process in a subprocess activity solely for the purpose of defining that event context. For example, if in an order handling process having steps A to Z, the customer may cancel or change the order between steps B and G with a particular handling flow, the modeler may draw a subprocess enclosing the portion from B to G and attach a message event to it. Order cancellation or change downstream, having a different consequence and handling flow, could have other message events placed there. In this way, attached message events provide a business-friendly notation for explicitly indicating both the scope and handling of external events.

An *error event*, drawn with a sine wave icon inside, indicates an error condition in the process. An error end event means the process throws the signal when it reaches the end event. An explicitly thrown error signal is meant to be caught by an error event attached to a subprocess enclosing the error end event. (For this reason, there is no error start event.) It is also common to see an attached error event where no throwing error end event is drawn. This implicit throw usually indicates a *system fault* in the activity.

A *none event*, drawn with no symbol inside, is by far the most common. A none start event means the trigger is unspecified. A diagram with multiple none start events is ambiguous. Does this mean all of the none starts are simultaneously enabled, or not? It should be avoided. In a subprocess, you always use a none start, since the trigger is the sequence flow into the subprocess.

None end events simply mean no signal is thrown when the end event is reached. They are very common. A none intermediate event in sequence flow indicates a named state or milestone in the process. It is rarely seen, but actually mirrors similar semantics in other modeling notations. An attached none event is not allowed.

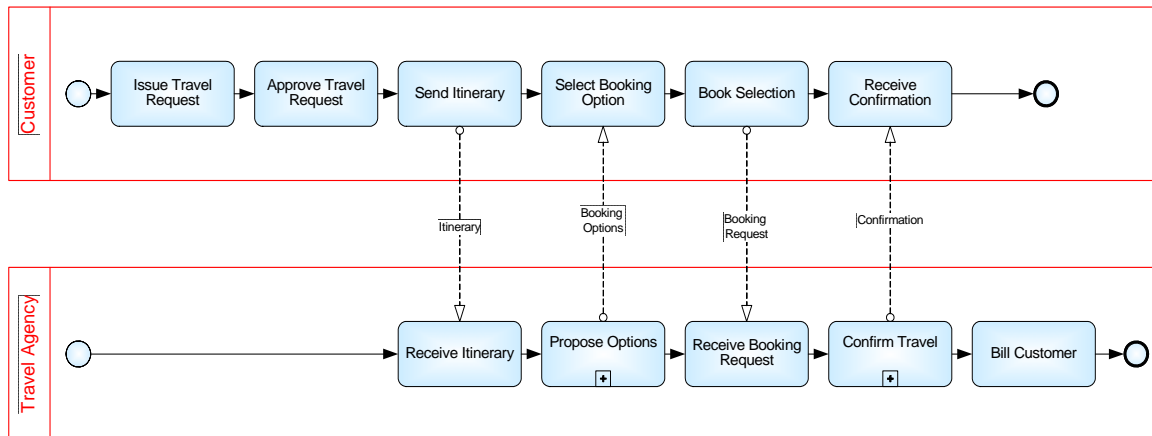
There are other event types, which will be described in part 4 of this series, but these are the important ones to know.

## Pools and Choreography

So far we have been discussing what BPMN calls *orchestration*, i.e., one entity's side of what might be a multi-party process linking, say, a buyer to one or more sellers and middlemen. An orchestration has a well-defined beginning and end and describes the sequence of steps within it as if there were some "engine" that has visibility and control over

it. That control does not extend to the external entities, however. Interactions between the orchestration and the other parties to the global process must be expressed as signals, often requests and responses, which BPMN calls *messages*.

BPMN has a way to represent the message exchange patterns linking orchestrations, which it calls *choreography*, using a dotted line arrow called a *message flow*. The business process diagram requires there for a container for each orchestration, called a *pool*. If there is no need to show choreography in the diagram and there is only one process, often the pool is implicit, not drawn.



**Figure 9. Orchestration is shown within a single pool. Message flows between pools indicate the choreography.**

Message flows can only be drawn between pools, not within a pool. Choreography mostly provides business context in BPMN models, not executable semantics. (The executable semantics in the orchestration is shown by the message events or send and receive tasks that generate and act on message events.) In many diagrams, choreography is not shown at all. Also, there is considerable evidence that OMG is significantly modifying the choreography notation of BPMN. However, some tools do make use of message flows to model executable semantics, such as partnerLinks in BPEL.

## Lanes and Artifacts

The rest of the BPMN notation represents second-class elements, things that give diagrams business-meaningful context but have no precisely defined semantics, validation rules, or mapping to executable implementation. This is surprising to many people. What about swimlanes? Data flow? Both second class, in the sense that each tool vendor defines its own usage of these concepts.

*Lanes* are subdivisions of a pool, typically representing organizational boundaries or human task roles. Technically they can mean whatever you want them to mean, and BPMN has no rules about what can or cannot cross a lane boundary.

*Data objects* are a type of BPMN “artifact,” meaning having no standardized semantics. A data object can represent a business object, a document, or anything else for that matter. It can be linked to a sequence flow (or message flow) by a dotted line called an *association*, and data flow to or from activities can be shown by associations drawn with arrowheads. But the data flow has no defined control over the orchestration.

*Annotations* are text comments in the diagram, linked to elements via association. A *group* is a dotted line box enclosing any part of the diagram; it has no defined semantics whatsoever.

And that's all of BPMN! Next time we'll look at the art of modeling with BPMN, including some methodology and best practices.

*Bruce Silver*