Stuff happens.

Boundary events help us describe how to handle errors, escalations, and time-outs that require deviations from the happy path.

Consider this order processing flow. The process is instantiated upon the receipt of an order. The order is processed and shipped. Once the order has been shipped, the instance of the process is complete.

But what happens if the customer cancels the order before it's shipped? Boundary events help manage such exceptions. In this case, when an order is canceled, the Process order activity is interrupted, the token is re-routed to the Update CRM activity, and the process instance is complete.

Boundary events are always intermediate events. Furthermore, boundary events are always catching event types. That is, they trigger other sequence flows, activities, and events when a specified exception occurs. The most common boundary events are error, escalation, message, and timer events.

Internal exceptions are caused by something that happens within the activity with which the boundary event is associated. Internal exceptions trigger error and escalation boundary events. External exceptions trigger message and timer boundary events.

Boundary events are further categorized as being interrupting or non-interrupting. Interrupting boundary events have borders consisting of solid, double lines. When activated, interrupting boundary events will—as their name suggests—interrupt the activity with which it is associated.

In the previous example, the Order canceled boundary event is an interrupting message event. Upon receiving a cancellation message from the customer, the activity is stopped, and the process flow is diverted.

Let's look at other examples of interrupting boundary events, starting with the interrupting error event.

Imagine you are a retail store. Your inventory replenishment process is instantiated when the stock level of an item falls below some predetermined minimum. The happy path includes the Replenish inventory sub-process, shown here in its expanded state. If the item is unavailable, an intermediate throwing error event is triggered within the sub-process. The corresponding error boundary event catches the error and interrupts the sub-process. Next, the unavailable item is removed from the catalog, and the instance of the process ends. Here's what the token flow looks like.

While an error event indicates that something has gone wrong within an activity or process, an escalation event is used when some threshold of time, cost, or effort is reached that requires some remedial action.

For example, consider this customer support process. Upon receiving a support request, a Customer Support Representative will endeavor to resolve it. If the CSR can't resolve the problem to the customer's satisfaction, an interrupting escalation boundary event is triggered. At this point, the CSR's task is interrupted, and responsibility for resolving the support request is transferred to a manager.

Activities can also time out, which is where the interrupting timer boundary event comes into play. For example, consider this article publication process. The process is instantiated every Monday. The happy path consists of two activities: the Select topic task and the Publish article sub-process, shown in expanded form. If the Publish article sub-process hasn't been completed by the deadline on Friday, the interrupting timer boundary event is triggered, and the process instance ends.

Non-interrupting boundary events have borders consisting of double, dashed lines. When triggered, a non-interrupting boundary event will initiate a new sequence flow but won't stop the activity with which the boundary event is associated. Let's take a look at an example of a non-interrupting message boundary event.

This is a variation of the order processing flow introduced earlier. The boundary event is triggered when a change of address is received from the customer while the order is being processed. The boundary event creates a second token that flows to the Update CRM activity and is consumed by the Address updated end event. At the same time, the Process order activity continues unabated, and the original token flows in due course to the Ship order activity and the Order processed end event, where it is consumed. The process instance ends when all tokens have been consumed.

The terminate end event is sometimes used in conjunction with boundary events because they both are related to the handling of exceptions.

Consider an order fulfillment process in which a customer inadvertently enters invalid payment information. In this case, you would want to prompt the customer to re-enter their payment information to complete the fulfillment process successfully.

However, if the fraud likelihood score exceeds some threshold, you may wish to cancel the transaction. The terminate end event clarifies that the entire process has been canceled, not just certain activities.