Sending Persistent State Signals

The following chapters describe how to send and handle signals of persistent state objects.

After an object has been created, the only way to influence its behavior is to send signals targeted at this object. This can either be done by using the <<PersistentStateSignalAction>> or the <<PersistentStateBroadcastSignal>> adapter. Both send signals, either targeted at a certain object given by its handle or to a whole set of objects defined by search predicates

Upon sending a signal, a correlation ID will be logged to the transaction log (see Contents of the Transaction Log). If a conversation ID is supplied, this will be the conversation ID. Otherwise, a unique value will be generated.
The same correlation ID will be logged for the send action and for the triggered transition.

On this Page:

- Sending Signals
- Broadcasting Signals
- Handling Undeliverable Signals

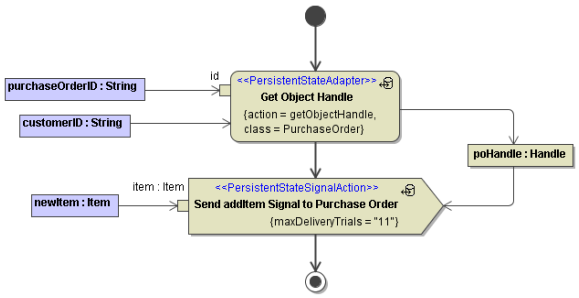
Related Pages:

- Handling Persistent State Objects With the Persistent State Adapter
- Contents of the Transaction Log

Sending Signals

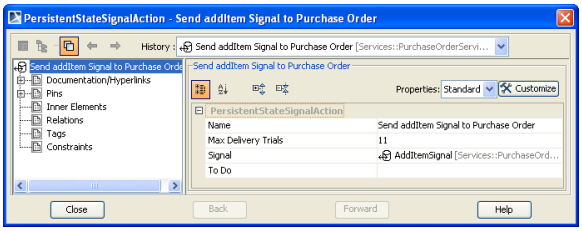
A handle can be used to send signals to the object instance represented by the handle. To send a signal the <<PersistentStateSignalAction>> stereotype is used.

*Figure: Sending Signals (Activity Diagram **Add New Item**)*



The send signal action requires a valid **handle** referencing an existing persistent state object. The type of signal to be sent – in this case **AddItemSignal** – is defined in the specification dialog, as shown below. Attributes declared in the chosen signal classifier and mappings in the activity diagram define the **signal parameters** that can be sent (see figure above).

Figure: Configuration of Persistent State Signal Action



If the signal cannot be delivered, the xUML Runtime calls the default handler for the sent signal as specified in the target class. For more details about this mechanism, refer to the next chapter.

The tagged value **maxDeliveryTrials** (also depicted in the above figure) is deprecated since it does not comply with the UML semantics of sending signals.

If errors occur in the context of the activity diagram, they must be handled. Otherwise, all persistent state actions are rolled back. However, there is the possibility to define a default error handler that is invoked for all unhandled errors (see Handling Unhandled Errors).

See Signal Events for more information about how signals are processed on the receiving side.

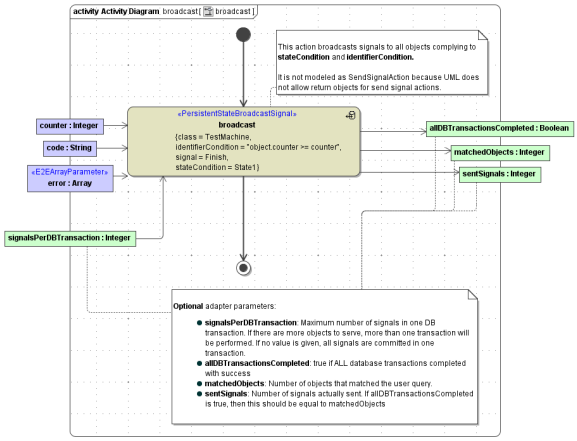
Broadcasting Signals

Example File (Builder project Advanced Modeling/PState):

<your example path>\Advanced Modeling\PState\uml\pstateBroadcast.xml

If signals are to be sent to more than one object it is possible to fetch a set of handles using `getObjects Handles` and then send signals to each of these objects by iterating over the handles. However, this is rather inefficient, thus we recommend using the <<PersistentStateBroadcastSignal>> adapter for this purpose. This adapter sends a signal to a set of objects that is defined by the tagged values **stateConditi on** and **identifierCondition** . The semantics of these tagged values is the same as for getObjectsHandles . In the example below, we send the signal **Finish** to all objects of class **TestMachine** being in state **Stat e1** having a **counter** attribute bigger than the counter object:

Figure: Broadcast Signals (Activity Diagram Add New Item)



The input of the <<PersistentStateBroadcastSignal>> adapter consists of the following objects:

- Objects used in the **identifierCondition** : the object **counter** is used in the **identifierCondition** t o filter all PS objects having a counter greater than the counter given in the activity diagram.
- Members of the signal given by the tag **signal** : the objects **code** and **error** are members of the signal **Finish**.
- **signalsPerDBTransaction:Integer :** this optional input parameter defines the maximum number of signals in a DB transaction. If there are more target objects, more than one transaction will be performed. If no value is given, all signals are committed in one transaction.

The output of the <<PersistentStateBroadcastSignal>> adapter consists of the following objects:

- **allDBTransactionsCompleted:Boolean :** this optional parameter is true if all database transactions completed successfully.
- **matchedObjects:Integer:** optional parameter giving the number of objects that matched the user query.
- **sentSignals:Integer:** optional parameter giving the number of signals actually sent. If **allDBTra nsactionsCompleted** is true, this should be equal to **matchedObjects.**

Handling Undeliverable Signals

When sending signals, the xUML Runtime puts them into an event queue where they stay until being delivered. However, it might happen that a signal cannot be delivered, because the target object is not in a state that accepts this very signal.
The xUML Runtime will try to deliver the signal once. If a signal cannot be delivered, the default signal handler is invoked. Signals that do not have a default signal handler will be silently discarded. Thus, it is strongly advisable for the modeler to use the default handler mechanism.
In the PurchaseOrder example, the persistent state class defines an operation **closeSignalDefault** (see figure *A Persistent State Class*). For the use as a signal handler, this operation requires the stereotype < <PersistentSignalDefaultHandler>> and has to be defined as **static**. As parameters, it takes the undelivered **signal** and a structure of type **Event**. While signal is defined model specific, **Event** holds generic system information about the signal delivery. The most important attributes are:

Element	Description

id	is a unique identification of the event.
classifier	is a reference to the signal class.
objectID	is the unique identifier of the object the signal belongs to, this is identical to the objectID of an object handler.
eventType	in this case is **Signal**.
eventBlob	holds the content of the signal. Do not use this attribute, as the same content is decoded into the signal parameter.
eventTS	holds the planned time of event delivery. In this case, it is the time of the last delivery trial.
creationTS	holds the time when the event was created. In this case, it is the time when the signal was sent to the object.

For example, the activity diagram below implements the default handler operation of the **CloseSignal** for the **PurchaseOrder** class. This means, if the **CloseSignal** cannot be delivered, this activity diagram **clos eSignalDefaultHandler** will be invoked. The activity diagram gets the undeliverable signal **closeSignal** (not used here) and an object of type Event with system information as input.

Figure: CloseSignal and its Default Handler Implementation (Activity Diagram Default Close Handler)