Handling Persistent State Objects With the Persistent State Adapter

The following chapters describe how to create persistent state objects, retrieve persistent state object handles, and get copies of persistent state objects. For all these operations, actions of stereotype <<PersistentStateAdapter>> are used.

The main tagged value provided by the <<PersistentStateAdapter>> stereotype used throughout this chapter is **action**. It describes the adapter functionality to be invoked. Currently the following actions are supported:

- create
- getObjectHandle
- getObjectHandles
- getObjectCopy
- loadExternals
- getObjectState
- commit
- rollback

After an object has been created, the only way to influence its behavior is to send signals targeted at this object. These actions are described in detail on Sending Persistent State Signals.

On this Page:

Related Pages:

Sending Persistent State Signals

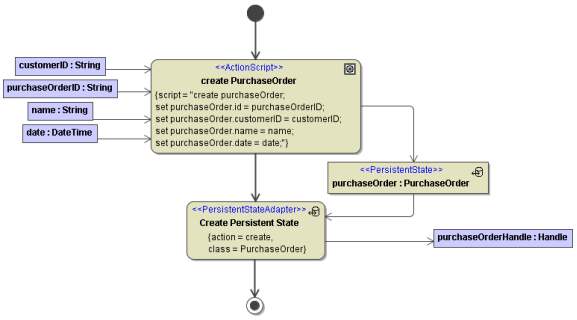
Creating Persistent State Objects

To create a persisted instance of an object, an instance of the stereotype <<PersistentStateAdapter>> with **action** set to **create** is used.
The compulsory tagged value **class** defines the type of the created persistent state object. This class always has to be of stereotype <<PersistentState>>. A reference to it will also be stored in the attribute **classifier** of the returned handle.
There are two ways to pass the initial values of the created object:

- If only the primary key values are known, these are passed as separate input values to the persistent state adapter. In this case, the persisted object will be empty except for the primary key fields.
- Alternatively, a pre-filled instance of the given class can be passed. In this case, the persisted object will hold a copy of the given input.

In both cases, at least the primary key attributes have to be provided. The resulting object has to be unique in the state database. Otherwise, the object cannot be created and an error will be thrown.
The effect of this action is that the transition from the objects initial state is executed **synchronously**. For example, in the **PurchaseOrder** state machine (see figure *State Machine Diagram of a Purchase Order*), the transition from the initial state is triggered and the event handler **Initialize Handler** is invoked. If an error occurs while processing the initial transition, the creation of the object will be rolled back and an exception is thrown.
After the **create** action, the object is ready to receive signals that trigger state transitions.
The **create** action returns an instance of the class **Handle** which is a reference to the created object. This handle can be used to send signals to or get copies of a persistent state object.

*Figure: Creating a Persistent State Object (Activity Diagram **Create Purchase Order**)*



If errors occur in the context of the activity diagram, they must be handled, otherwise, all persistent state actions, including **create**, are rolled back.

Related Error Codes

Find a list of all persistent state error codes on System Errors of the Persistent State Adapter.

Error Code	Description
PSADSM/5	A persistent object with the same primary key already exists.

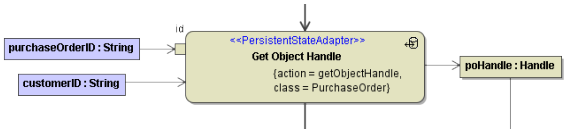
Retrieving Object Handle

After creation, an object handle represents a persistent state object. With these handles, it is possible to retrieve information about and to send signals to the represented objects.
While a handle is returned by the create action and can be passed as parameter in and out of operations and services, it is also possible to retrieve a handle for an object specified by its primary key and class.
A <<PersistentStateAdapter>> instance with **action** set to getObjectHandle is used to implement this functionality.
The compulsory tagged value class defines what kind of object to look for. This class always has to be of stereotype <<PersistentState>>. A reference to it will also be stored in the attribute classifier of the returned handle.
As input, the persistent state adapter requires all primary keys as defined in the given class. In our example, the attributes id and customerID have the stereotype PrimaryKey and identify a persistent state object unambiguously (see class diagram in figure *A Persistent State Class*). The input object flow states **purchaseOrderID** respectively **customerID** are mapped to these attributes.

> If the result set is not unique, or no persistent state object could be found, an error will be thrown that should be handled. This is not done in this example. For more information about error handling, see section Modeling Error Handling.

If no error occurs, the result is returned in an object of type **Handle**.

*Figure: Action State **getObjectHandle** (Activity Diagram **Get Purchase Order**)*



> The object state **poHandle** is of complex type **Handle**. This class is located in the Persistent State module in package **Data / Persistent State / Services / Objects**.

Related Error Codes

Find a list of all persistent state error codes on System Errors of the Persistent State Adapter.

Error Code	Description
PSADSM/12	Requested persistent object does not exist.

Retrieving Multiple Object Handles

An object handle represents a single persistent object instance. The previous explains how to do this for exactly one handle. However, it is frequently necessary to retrieve sets of object handles that fulfill certain conditions.
This can be achieved by using the <<PersistentStateAdapter>> and the **getObjectHandles** action. The result of this action is an array of handles that corresponds to persistent objects of type **class** that comply with the criteria given in the **identifierCondition**.

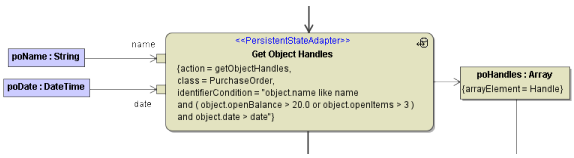
> If no persistent state objects could be found, **no** error will be thrown. The array of handles will be empty.

Within the identifier condition, one can access object identifier attributes by the keyword **object**. The referenced object attributes must be stereotyped as **SearchKey**. All other variables used in the identifier condition statement must be given as input of the action state:

- If there is no input mapping defined – e.g. by the use of input pins – the input in the identifier condition is given by the input object flow states.
- If there is an input mapping, the input is given by the input parameters such as **date** as defined on tab **Pins** on the action specification dialog.

In the example below, all open purchase orders, for which the identifier conditions evaluates to true, are retrieved. These purchase orders are stored in the array **poHandles**.

*Figure: Retrieving Handles to Multiple Persistent Objects (Activity Diagram **Add Gratifications**)*



> The array elements of the output object flow state **poHandles** must be of type **Handle** (the **Handle** class is located in the Persistent State module in package **Persistent State / Services / Objects**).

Besides using identifier conditions, it is also possible to have a state condition to select objects being in a given state. This is done using the **stateCondition** tagged value. It refers to a given state. An example can be found in the activity diagram **Get Purchase Orders By Date** described in Getting Copies of Persistent State Objects.

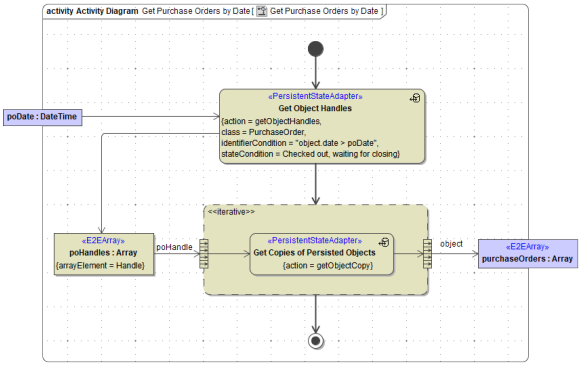
Getting Copies of Persistent State Objects

The <<PersistentStateAdapter>> action **getObjectCopy** returns a **copy** of a given persisted object. As input, the action requires an object handle.

> Please note, that the output of this adapter action, even though an instance of the persistent state class, is only a **snapshot** of the persisted object. Any changes to the returned object or to the persisted objects will have no influence on each other.

Runtime 2019.10 Builder 7.6.0 If you have defined the persistent state class as to have external attributes (see Persistent State Classes > External Persistent Data), these attributes will only be loaded to the object copy, if you set tag **withExternals** to true on the adapter action.

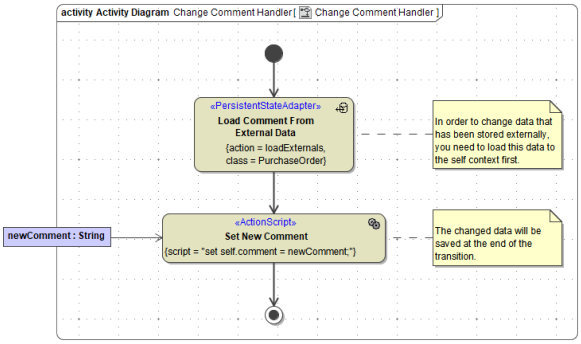
In the following example, a list of object handles to **PurchaseOrder**s newer than a given date and in the state "Checked out, waiting for closing" is retrieved. Then it iterates over the array of object handles to retrieve a copy of the content of each **PurchaseOrder** object.



Loading External Persistent State Data

Runtime 2019.10 Builder 7.6.0 You can define persistent state attributes as to be external (see Persistent State Classes > External Persistent Data), so their content will only be loaded on demand. This applies to getObjectCopy as well as to the self context within persistent state transactions.

To load the externally stored data in self context, you need to call the <<PersistentStateAdapter>> with action **loadExternals**.



If you have loaded the external data once in a persistent state transaction, it will be considered loaded by the xUML Runtime. Subsequent calls of **loadExternals** will be omitted and logged to the service log (log level DEBUG). This is to prevent modelers to accidentally overwrite changes to the persistent state object.

> If you change external attributes in self context, they will be stored at the end of the transaction. This applies only if they have been loaded before.

Getting State Info

> Deprecated This function is deprecated. Please use the Persistent State Control Adapter instead, esp. queryObjects().

The <<PersistentStateAdapter>> action **getStateInfo** returns information about the current state(s) of an object.

A persisted object can be in multiple states at once. The obvious case occurs when using fork to split the process flow into parallel execution. The other, non-obvious but more common case is the use of composite states. When inside a sub-flow, the persisted object actually is in two states: the current state inside the sub-flow and in the enclosing parent state.

Output of **getStateInfo** is an **Array** containing elements of type **StateInfo**:

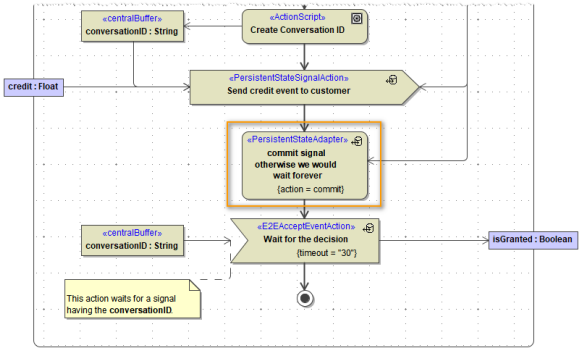
Attribute	Description	Example
id	Name of the state.	`Purchase_order_is_initialized__waiting_for_further_orders`
type	Type of the state, see States for more information.	One of: `STATE`, `FORK`, `JOIN`, `CHOICE`, `ENTRY`, `EXIT`, `COMPOSITE`, `SUBMACHINE`, `ORTHOGONAL`, `HISTORY`, `INITIAL`
entryTime	Time object entered the state.	`2019-09-16T09:00:06.0Z`
token	Internal id.	
parentToken	Internal id.	

Committing Changes to the Persistent State Database

Because of the transactional nature of bridge requests, changes to persistent state objects are not written to the persistent state database right away. Until such changes are committed, they are kept in memory only. Normally and if no errors occur, changes are committed at the end of a Bridge request (e.g. a service call). As soon as the database is updated, the changes to the persistent state object (including regular or conversational signals) are available for the persistent state engine to process them.

Sometimes there is a need to update the database (and trigger the persistent state engine) somewhere in the middle of of a request (e.g. for persistent state conversations, see Conversations). This can be done using the **commit** action on the <<PersistentStateAdapter>>.

Figure: Committing Changes to the Persistent State Database



Rolling Back Changes to Persistent State Objects

Because of the transactional nature of bridge requests, changes to persistent state objects (including regular or conversational signals) are not written to the persistent state database right away. Until such changes are committed (see also Committing Changes to the Persistent State Database), they are kept in memory only. If an error occurs while working on a persistent state object, any changes that are not yet committed will be rolled back automatically. This rollback can be triggered manually by calling the <<PersistentStateAdapter>> with action **rollback**.

For more details see also Persistent State Transaction Concept.