Persistent State Classes

Each class of which instances have to be persistent must have the stereotype <<PersistentState>>. One or more attributes of this class must have the stereotype <<PrimaryKey>>, uniquely identifying an object. A persistent state class optionally has attributes stereotyped <<SearchKey>>.

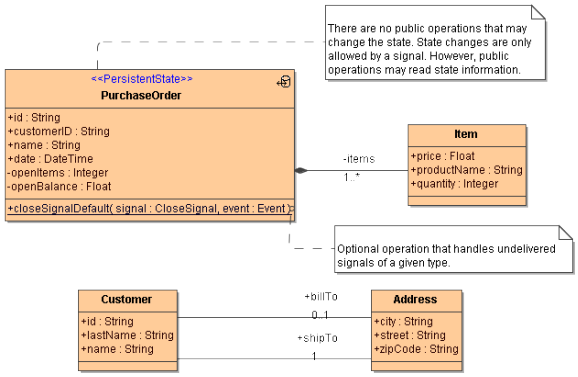
On this Page:

Primary Key and Search Key Attributes

While primary keys are used to identify and address a specific unique object, search keys are used to query for a list of objects. If you want to query for a list using primary key fields, these fields also require setting the <<SearchKey>> stereotype.
Only attributes of primitive types can be marked as primary or search key.

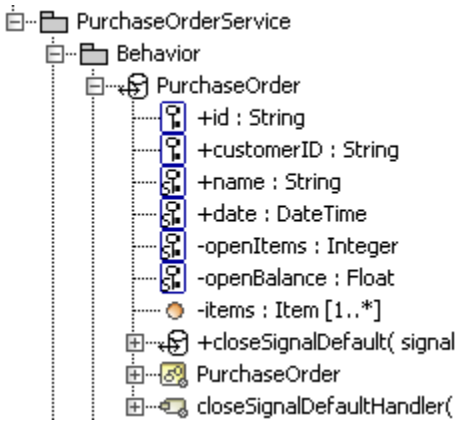
> Search keys that have not been initialized are `NULL`. Keep this in mind, if you want to look-up all persistent state objects using `mySearchKey like '%'`. Either, you need to initialize the search key, or you need to extend the search phrase to `mySearchKey like '%' or mySearchKey is NULL`.

Figure: A Persistent State Class



Attributes having stereotype **PrimaryKey** or **SearchKey** are represented with different icons in the containment tree of MagicDraw (see picture below).

Figure: Stereotyped Attributes of a Persistent State Class



Persistent Data

Attributes that are not marked as primary key or search key contain persistent data.

Private Persistent Data

Builder 7.5.0 You can mark attributes of a persistent state class as private by applying stereotype <<E2E Private>>, but we do not recommend this.

Attributes marked as private are not being serialized by the xUML Runtime. As the Runtime cannot distinct the context of <<E2EPrivate>>, this results in such attributes not being saved between transitions and states.

Refer to Hiding Attributes From Interfaces for more information.

External Persistent Data

Runtime 2019.10 Builder 7.6.0 You can mark attributes of a persistent state class as external by applying stereotype <<External>>. This can speed-up persistent state performance if you have huge data objects (like e.g. big blobs, IDocs or PDFs) that are only used in few transitions.

External persistent data will be handled differently compared to the internal data as comes to the following:

- **External persistent data will be stored separately.**
  By specifying a divergent alias in the Persistent State Components you can even store external data to a different database.
- **External persistent data will only be loaded on demand.**
  getObjectCopy() will only load external persistent data if tag **withExternals** is set to true. In self context, you need to load external persistent data with a persistent state adapter action with action **loadExternals**.
- **External persistent data will only be saved if loaded before.**
  Changed external persistent data will be saved at the end of a persistent state transaction, but you need to load them before. So the correct procedure is:
  1. Load external persistent data (even if they are empty).
  2. Apply changes.
  3. Saving will be done automatically at the end of the transaction.

  See also the example at Handling Persistent State Objects With the Persistent State Adapter > Loading External Persistent State Data.

Using external persistent state attributes, you need to respect the following:

- Stereotype <<External>> can only be applied to strings, blobs, arrays and classes.
- External attributes cannot be a primary or a search key, so do not mix stereotypes <<External>>, <<PrimaryKey>> and <<SearchKey>>.

Modifying Persistent State Classes

Be careful with modifying persistent state classes of a service already running on a Bridge. If you deploy the modified service, the service might refuse to start-up and throw an error:

```
Info: Loaded addon PState_AddOn successfully.
Fatal: Found persistent state objects of class "<name of the old class>" that
does not exist in the repository anymore.
Fatal: Detected objects for which no metadata exist in the repository.
Correct the problem either by restoring class definitions in the model or by
deleting corresponding objects.
Fatal: Initializing service "PersistentStateService" failed
```

This can happen e.g. if you remove or change the name of the persistent state class but the persistent state database still contains objects of the old class. These objects become orphaned in this case.
You can fix this in two ways:

- Delete all old persistent state objects using the Bridge (see Deleting Persistent State Objects).
- Redeploy the old service and let it work all persistent state objects before deploying the new one.