Examples

Three examples of defining object names, with a unique local name (within a container), a repeated local name (across containers), and a subcomponent name, are given below.

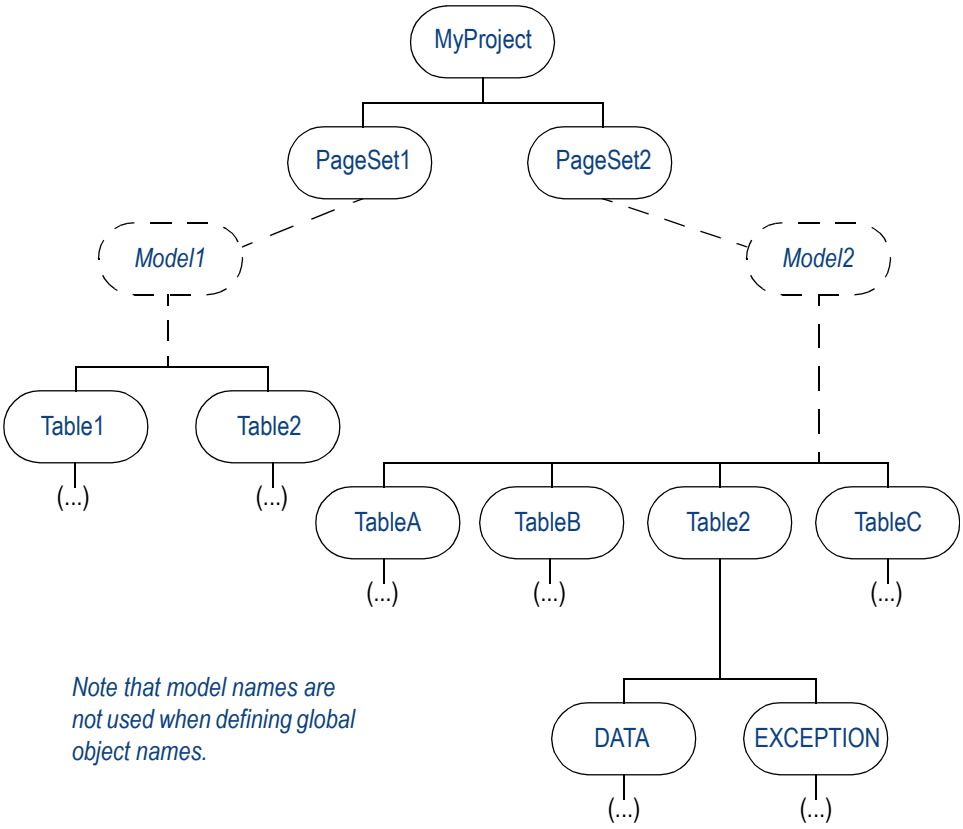

Figure 8 *Defining Composer object names (example)*

Example 1: Table object name (unique table name)

To find the global name of the Composer object "TableB" in Figure 8, you do the following:

1. Retrieve the project object name and contents.
   Name = "MyProject"
2. Retrieve the model object contents.
   Name = "MyProject"
3. Retrieve the page set object name(s) and contents.
   Name = "MyProject.PageSet2"
4. Retrieve the table object name(s) and contents.
   Name = "MyProject.PageSet2.TableB"

Example 2: Table object name (duplicate table name)

To find the global name of all Composer objects called "Table2" in Figure 8, you follow the same process as in Example 1 above, following the hierarchy for each object separately.

The global object names for each are "*MyProject.PageSet1.Table2*" and "*MyProject.PageSet2.Table2*" (note that though the local table names are the same, the global object name for each is unique).

Example 3: Subcomponent object name

To find the global name of a cell contained within:

- Project "MyProject" (see Figure 8)
- Page Set "PageSet2"
- Table "Table2"
- Row Set "DATA"
- Row 4 (as defined by the DIS_rownum column in the Interface table)
- Column "PRICE"

you follow the same process as in Example 1 above, using the conventions for subcomponent names as described in "Exceptions and special cases" on page 31.

The global name of the specified cell subcomponent in "MyProject" would be *MyProject.PageSet2.Table2.DATA.4.PRICE*.

Object spaces

When working with scripting and the Repository, you make changes to the contents of the Repository by using functions called on the APIs. The Repository objects are the actual Composer data; you make changes to this data through the API functions.

Composer/Repository objects

The Repository contains the data objects (model sets, models, tables, etcetera) that make up the configuration data for Composer projects. The contents of the Repository directly relate to the contents of the Composer project: if a table exists in the Repository, then it exists in the Composer project. You affect the objects stored in the Repository using functions in Data Access API to perform operations.

API (application) objects

To perform these actions, you create a *representation* of the Repository object in the running application; this representation is called an *API object*. These API objects are viewports, or *handles*, into the real objects in the Repository, and are used to manipulate the contents of the Repository objects. Using functions on these handles, you can perform operations on the objects of the Repository (such as asking a model to delete a table). It is important to note that:

- Any operation on the Repository has an equivalent function call in the appropriate class in the ComposerObjects API. For example, the operation of removing a table from the Repository is mapped to the deleteTable function call in the ComposerModel object. It is only through calls to these functions that the Repository can be modified.
- Creation or deletion of a handle is unrelated to the creation or deletion of the corresponding object in the Repository (in other words, deleting a handle in the application does not imply that the corresponding object is deleted in the Repository). See “Memory Management” on page 38 for information about why you should regularly delete handles when you are finished with them.

If an API object exists in the running application (for instance, if there is a ComposerPageSet object in the application), it does not necessarily have to exist in the Repository (and will not exist in the Repository until the project is saved using the saveProject function.

Conversely, if an object does not exist in the application, it does not necessarily mean that the object does not exist in the Repository. For instance, if the application has asked for project P1 but not project P2, then no page set object from P2 has been created in the application (though the page sets for P1 are present). All page set objects in the project exist in the Repository, but only those whose containers have been requested exist in the application.

This distinction, between an API object and a Repository object, is important; it gives you the ability to perform operations on and request information about your Composer data within your running application (using the API objects you create), and then use API functions to commit any changes or updates to the Repository.