

4.2 ADM2 Terminology

Hopefully this section doesn't come too late. If you've been reading carefully, page by page, from the introduction to here, you may have already come across terms that aren't totally clear to you. To make sure the rest of this book makes some amount of sense, it becomes necessary to describe some of the common ADM2 terminology in greater detail. While not exactly the best topics for dinner conversation, at least we can be sure we are talking about the same things from here on out.

4.2.1 Method Libraries

Method libraries are groups of Progress include files that fulfill a specific role in the ADM2 architecture. It is these method library include files that make sure any needed SmartObject classes are instantiated in memory and added to the current procedure's SUPER stack for easy access. Following the hierarchical structure for classes, only the appropriate top-level method library is needed in each SmartObject master procedure. The top-level method library will drill down to the lower level classes one by one by including the next "lower level" method library. This is how the ADM makes sure each SmartObject class is properly instantiated such that the most basic class is on the bottom of the SUPER stack and above it are added the increasingly more specific SmartObject classes.

The method libraries are found wherever Progress is installed, in the *src\adm2* directory. Inside the method library, there are a few key areas that we'll examine closer. Let's take a look inside *src\adm2\viewer.i*, the SmartViewer method library to understand the basic construction used for these files.

Near the top of the method library, after Progress's copyright information, is the setting of viewer properties.

```
&IF "{&ADMClass}":U = "":U &THEN
    &GLOB ADMClass viewer
&ENDIF

&IF "{&ADMClass}":U = "viewer":U &THEN
    {src/adm2/viewprop.i}
&ENDIF
```

Figure 59 - Setting Viewer Properties

The *viewprop.i* include file does a number of things that don't affect most users of the ADM. It includes the *viewprto.i* include file which prototypes all of the internal entries in the current SmartObject class file. This is mostly to pave the way in case a developer wants to use non-Progress clients such as Java. Next, it includes the property file for the next most basic SmartObject class, and then adds its own instance properties to that list. Finally, any custom properties will be included.

After the *viewprop.i* include file is the drill down to the lower level method library for the data visualization class.

```
&ANALYZE-SUSPEND _UIB-CODE-BLOCK _CUSTOM _INCLUDED-LIB Method-Library

/* ***** Included-Libraries ***** */

    {src/adm2/datavis.i}

/* _UIB-CODE-BLOCK-END */

&ANALYZE-RESUME
```

Figure 60 - Including Lower Level Method Library

By recursively calling the lower level method libraries first, their associated SmartObject classes will be placed on the local procedure's SUPER stack earlier and the higher-level classes placed later on the SUPER stack. Since the stack is searched in a LIFO order, the "higher" (more specific) SmartObject classes will be searched first for any internal procedure or function call.

```
RUN start-super-proc("adm2/viewer.p":U).
```

Figure 61 - Instantiating The SmartObject Class

Next, we call a basic routine that looks for the indicated SmartObject class file already running persistent in memory. If the procedure is not found, it is first run persistently. Either way, its handle is obtained. The resulting handle is added as SUPER procedure to

the current procedure's local SUPER stack. All this is done by the *start-super-proc* internal procedure.

```
/* _ADM-CODE-BLOCK-START _CUSTOM _INCLUDED-LIB-CUSTOM CUSTOM */  
  
{src/adm2/custom/viewercustom.i}  
  
/* _ADM-CODE-BLOCK-END */
```

Figure 62 - Calling The Class Customization Include File

Finally, an include file controlling developer supplied class customizations is added. This causes any local SmartObject class customization files to be added to the SUPER stack on top of the default class, thus allowing the customized behavior to be found first, before any default behavior it is intended to overload.

4.2.2 Classes

SmartObject classes are Progress persistent procedures that contain a combined set of internal procedures and user-defined functions as well as some associated include files. Together, these pieces provide a standardized set of functionality that control and define the normal behavior specific to that class of SmartObjects. Each class fits into a hierarchy that goes from most general to most specific. The behavior of a given SmartObject class is inherited from the more general classes “below” it and passed on to the more specific classes “above” it in the SmartObject class hierarchy.

The illustration below graphically shows most of the basic SmartObject classes. The foundation of all of the others is the “smart” class. This class defines the most basic levels of SmartObject behavior with such functions as *getContainerHandle* and *getDataSource*. All of the other SmartObject classes build upon this behavior.

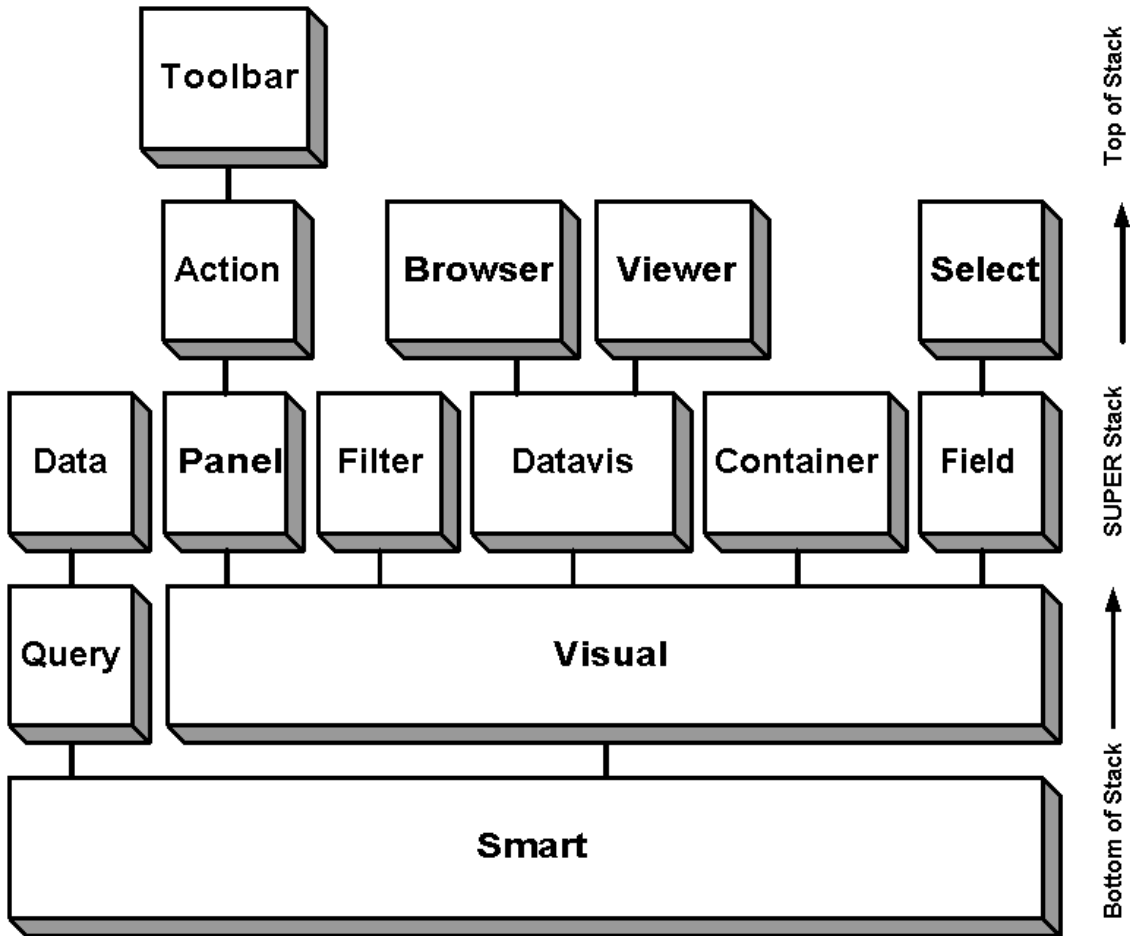


Figure 63 - Hierarchy Of Class Files

Each of the ADM classes is supported by standard and custom sets of class files. In most cases, the call to the custom class is commented out and thus ignored. It is only when a particular application needs to provide customization for an entire ADM class that the template provided by Progress is copied from its normal location in the Progress *src/adm2/custom* directory into a matching directory under the application code tree. The call to this class customization is then uncommented so that the custom class can be invoked into the SUPER stack after (and thus called before) the standard ADM class.

More information concerning the customization of SmartObject classes as well as specific examples can be found in later sections.

4.2.3 Templates

A SmartObject template is a baseline procedure file that the AppBuilder copies whenever a developer decides to create a new SmartObject procedure. Templates are normally found in the Progress *src/adm2/template* directory. To create new templates specific to

your company or current application, make a copy of the template file and place it in a matching directory in the application code tree. More details and specific examples for doing this are given in later sections.

The template file contains a call to the appropriate method library for that SmartObject. This is the basis for the whole structure of the ADM. They also contain code processed by the AppBuilder to control the design wizard that comes up whenever a new SmartObject of that type is created.

4.2.4 Masters

A SmartObject master is the end result of a programmer creating a new SmartObject in the AppBuilder, adding specifics, such as which SDO to reference and which fields to display, and saving it in their application directory. It starts its life as a template but ends up with all the blanks filled in, database tables and fields specified, visual aspects determined and layout finalized.

These are the files that the developer creates after saying “File – New” in the AppBuilder, giving the result a name and saving it. One SmartViewer master might be for the Customer table (via a SDO master for the customer table) but only show the name, customer number and address fields. It might contain local overrides for the *updateRecord* procedure from the data visualization class. Another SmartViewer master might use the same SDO but show other fields. Or show the same fields contain different overrides for modifying the default behavior.

4.2.5 Instances

An instance of a SmartObject is the persistent running of a specific SO master file, along with all of its runtime properties and its relations via SmartLinks to other SmartObjects in the container. Thus, a SmartObject instance is not a permanent thing that is saved to disk. Instead, it is the run-time combination of the specific details contained inside a SmartObject master and the circumstances of how that master procedure is invoked in one particular SmartContainer object.

If an SDO for the customer table, called *dCustomer.w*, is placed in one SmartWindow with a batch size of 500 records, that is one instance of that particular master file. If that same SDO is placed in another SmartWindow and given a batch size of 100 records, that is a second instance of the same master file. The first instance might be linked to one SmartViewer, while the second instance is being used to provide data to a SmartBrowser instead. It is the sum of all these run-time details that makes up what is referred to as a SmartObject instance.