Bentley Architecture

Custom BIM Object Creation

and the Bentley DataSet Files

By Tom MacKnight, Fluor Corporation

This paper provides a basic understanding of the methods for creating Custom Objects in the Bentley Architecture DataSet – how to create DataGroup Definitions and Catalogs as well as how the files work that make up the DataSet.

An Overview of Bentley Architecture BIM Objects

BIM Objects are representations of real world items. They represent a physical thing that can be seen or touched, for example, some material thing that occupies space. A BIM object can be cleaved into two parts: the *3D geometric* definition that describes its form and the *database* definition that conveys alpha-numeric descriptive information about the properties of the object. Object properties might include: name, size, weight, material properties, etc.

In Bentley Architecture (BA) there are three basic methods for creating the object definition: Compound Cell Manager, Parametric Frame Builder, and Parametric Cell Studio. Actually there are two other methods. One is to use a computer language to create and object, but this is not a likely method unless you are Bentley. The other method is to use a plain geometric line, shape, circle, surface, solid, cell, or other graphic entity and use the Add Instance Data command to attach data to the element to create a BIM Object.

This paper does not discuss the creation of an Object's geometric definition. Rather, the focus at hand is the database aspects of BIM Objects. The Bentley Architecture database is termed the *DataSet*. It is comprised of DataGroups and Catalogs. At a high level, creating a custom BA object in the database is a two step process. First an Object *DataGroup Definition* must be created, and then a *Catalog Type* can be created for the object. Of course, at a micro level each of these two steps requires many sub-steps to accomplish their creation.

The Object DataGroup Definition is like the skeleton or framework from which the information for Catalog entry can be hung. Another way to look at it is that the definition functions like pigeon-holes or mail-slots where information can be placed. Catalogs are collections of BIM Objects. A Catalog may simply consist of the data as defined in the Data Definition. Frequently it can point to the cell file that depicts the geometric definition. And, it often has dimensional information used to drive the size and shape of geometric definition. It may also have information driven from the geometric definition such as width, height, depth, area, and volume which is used to quantify the object.

In concept, an object definition could be used for more than one Catalog Type. As an example, Catalog Types for general architectural equipment, laboratory equipment, and kitchen equipment might all share common *equipment* DataGroup Definition. The definitions for laboratory and kitchen equipment can be extended with additional specific DataGroup Definitions for information unique to those types of equipment. This example points out another aspect of DataGroups, they can be used in additive fashion. More than a single DataGroup Definition can be used to define a Catalog Type – the mail slots can be glued together.

Creating an Object DataGroup Definition

The first step in creating a custom *DataSet* object for use in Bentley Architecture (BA) is to define the data structure for the object. This is done using the *DataGroup Definition Editor*. This is a standalone Windows application that can be started two way. From the Windows Start Menu via *Start*| *Bentley Building V8 XM*| *Building Tools*| *DataGroup*| *DataGroup Definition Editor*. It can also be started from within BA using the "dg defed" key-in. The dialog box appears as in Figure 1.

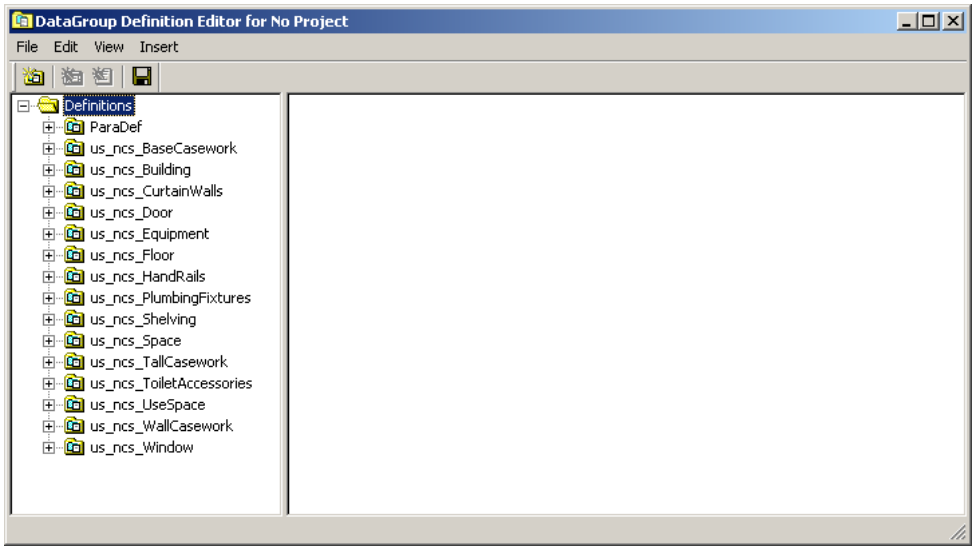

Figure 1 – DataGroup Definition Editor.

This application is used to create the definition for an object type. An object definition consists of a Definition Name with Properties and Property Groups. A new definition file is created one of two ways: 1) clicking on the New Definition File button, or 2) by right-clicking on the "Definitions" icon and selecting New Definition File option. This will bring up the New File dialog box which is used to give the definition a name. (See Figure 2)

New files are created in the directory folder as specified in the New File dialog box. The default folder for data definitions is *datagroupsystem*. It is not normally necessary to point to another location other than the default given. Once a name is entered and the OK button is clicked a definition file will be created in the given directory with the name entered and an extension of .xsd.

It is worth taking a few moments to talk about file naming conventions. Bentley has used the prefix "*us_ncs*", which stands for United States National CAD Standard. It is recommended that site specific prefix be used that reflects the company or project name. Here at Fluor we are using "*flr_*". Also note that in general, Bentley uses the convention of naming xsd files using first letter capitalized (e.g *us_nc_**C**urtain**W**alls* and not *us_nc_**c**urtain**w**alls*).

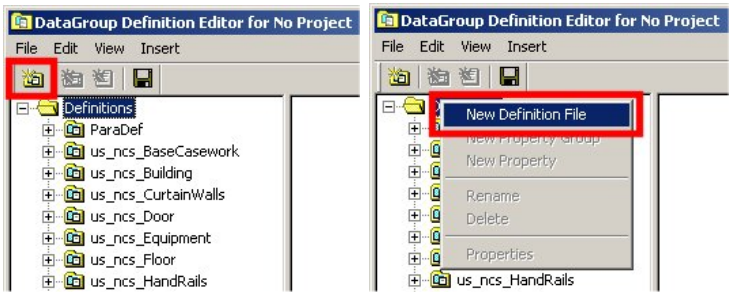

Figure 2 – New Definition File Creation.

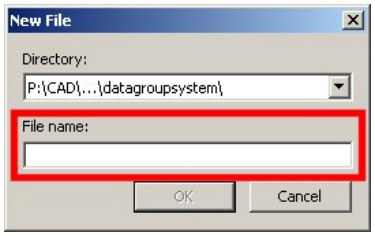

Figure 3 – New Definition File Dialog Box used to enter a name for the definition.

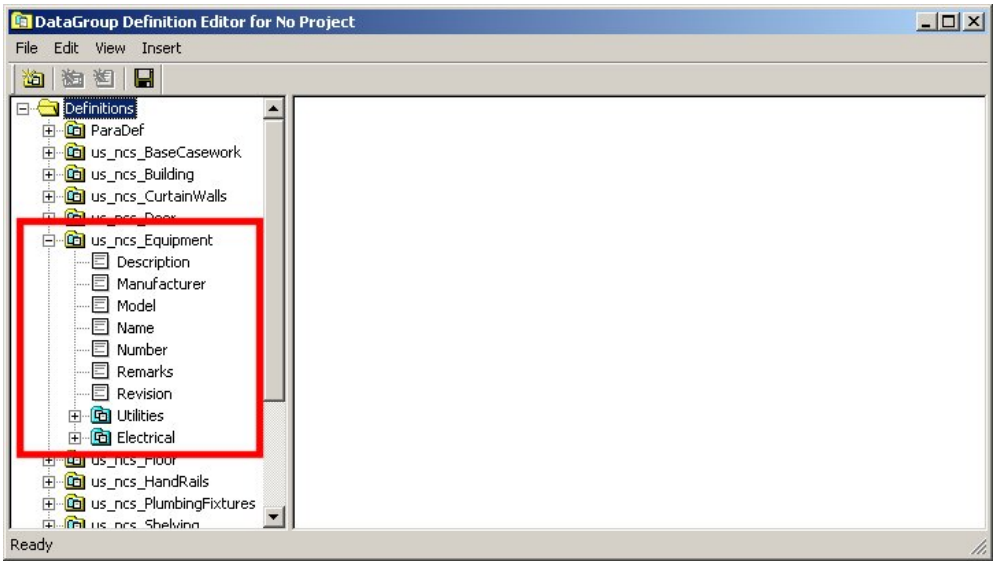

Figure 4 – Object Definition Tree expanded to see the individual Properties and Groups.

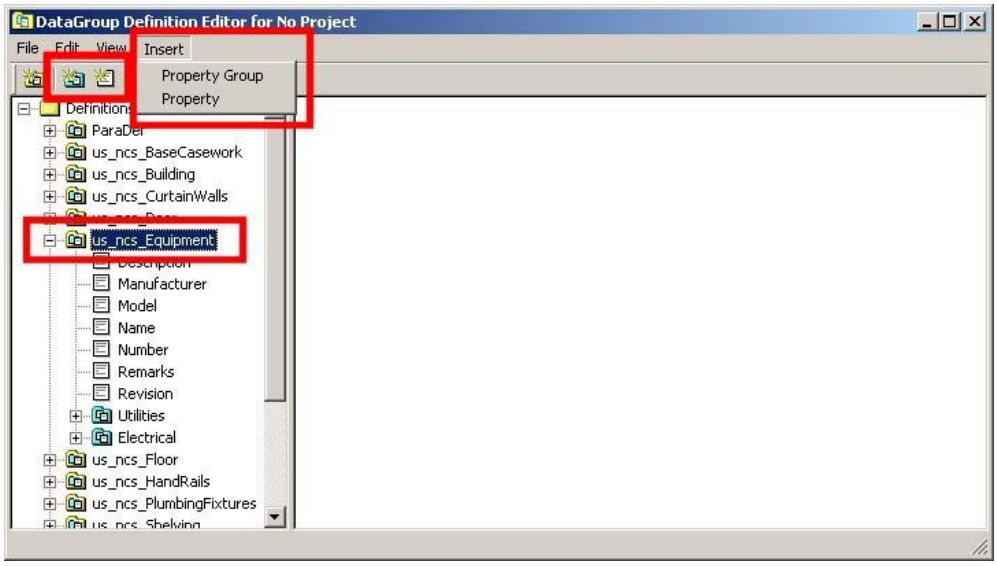

Figure 5 – Object Definition selected with active buttons and menu items

On selecting an object definition, the New Property Group and New Property Buttons are activated (non-grayed), and the Insert Property Group and Property drop-down selections are activated (non-grayed). (See Figure 5) Clicking on the appropriate button or selecting the appropriate menu item will create a new Property Group (1) or Property (2) in the definition tree. (See Figure 6)

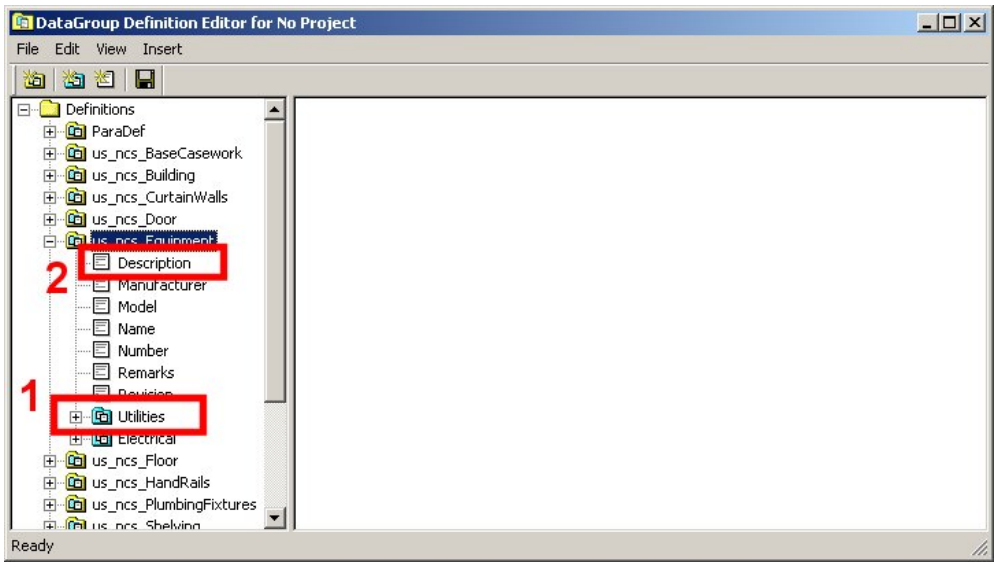

Figure 6 – Property Group (1) or Property (2).

Property Groups are simply collections of Properties. They may be nested under a definition or under other Property Groups. As shown in Figure 7, a Property Group named Electrical might contain a number of additional individual properties such as Volts, Amps, HP, Phase, etc.

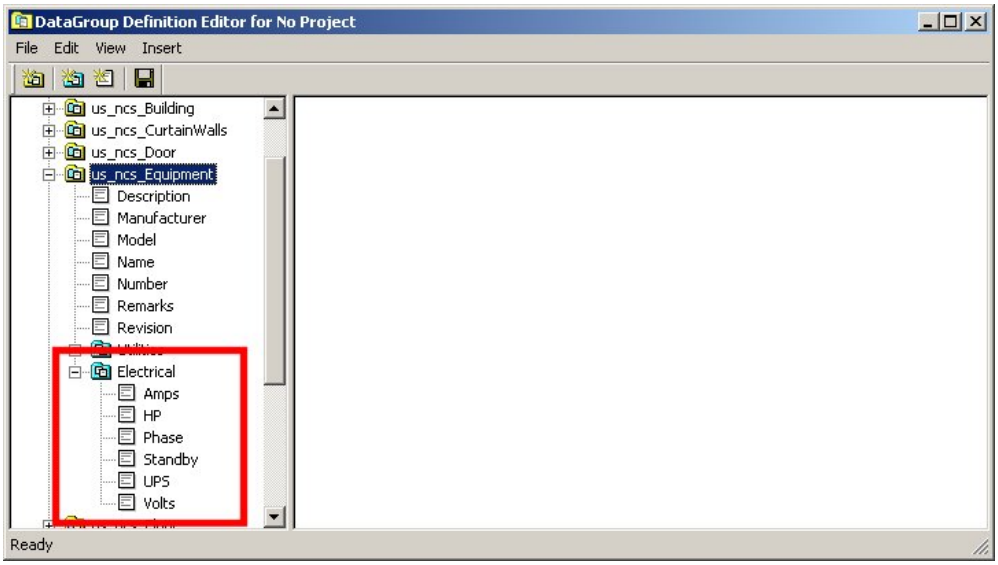

Figure 7 – Definition Tree with Property Group expanded.

When you add a new Property Group a text box is activated. Type in the name for the group and finish the entry with a tab or carriage-return. Alpha numeric characters (A-Z, a-z, 0-9) plus an underscore (_), dash (-), or period (.) are acceptable. Spaces and characters other than those given above are illegal. (See Figure 8)

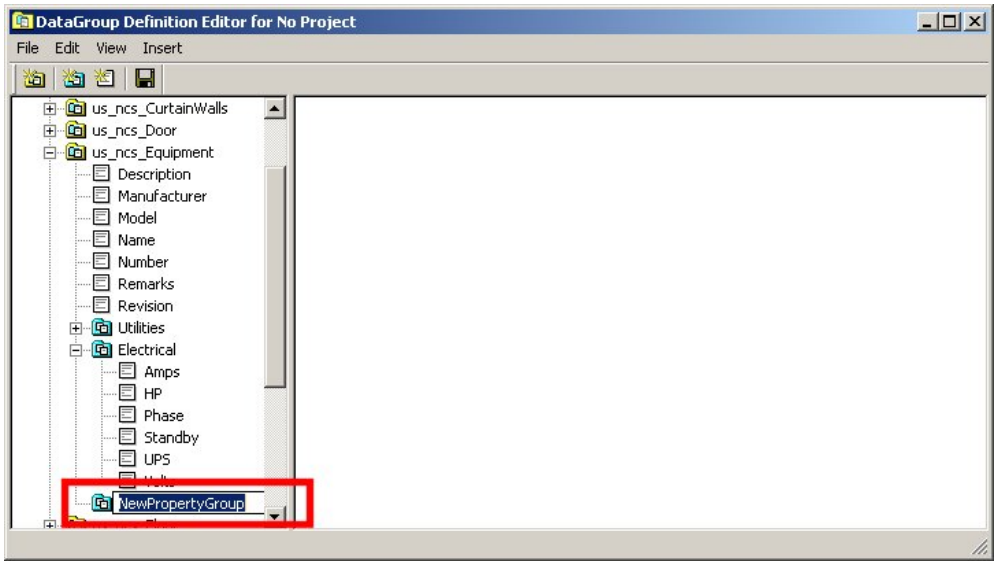

Figure 8 – Adding a new Property Group.

Similarly, when you add a new Property a text box is activated. Type in the name for the group and finish the entry with a tab or carriage-return. Alpha numeric characters (A-Z, a-z, 0-9) plus an underscore (_), dash (-), or period (.) are acceptable. Spaces and characters other than those given above are illegal. At the same time, the Property attributes appear in the right hand side of the editor. At a minimum you should enter a Display Name. (See Figure 9)

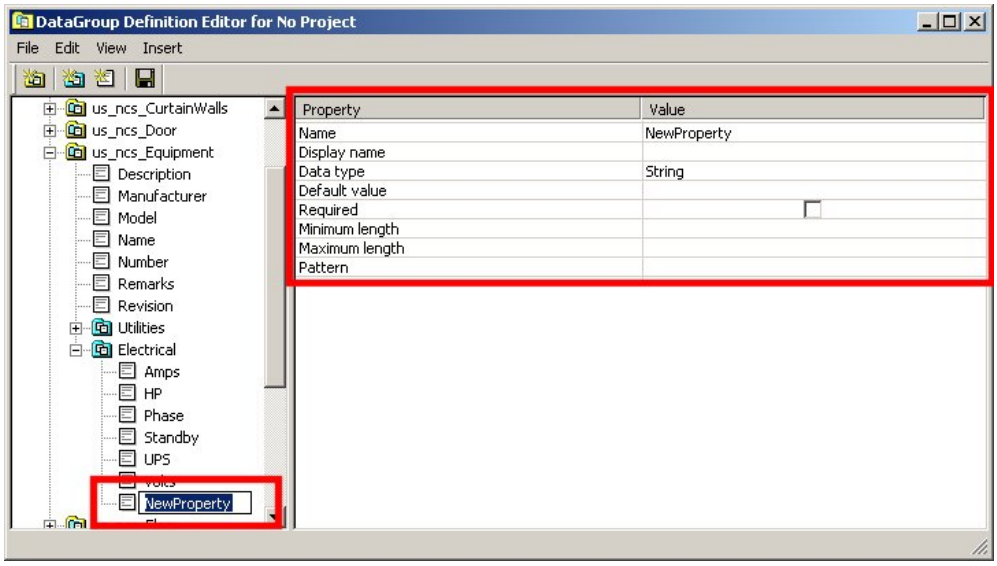

Figure 9 – Adding a new Property.

To rename or delete a Property Group or Property, right-click on the item and select the appropriate pop-up menu item. Once you have created the data definition for an object, you need to save the definition.

It should be noted that this similar process is used to add properties to existing DataGroup Definitions. For example, if we wanted to add a Cost Property Group with Material Cost and Labor Cost Properties to the DataGroup definition for a door, we would us the above process.

Creating a Catalog Type

Once a Object Definition has been created, a Catalog Type can be created. This is a relatively straightforward two-step process that involves creating an entry for the Catalog Type, and then marrying it with the Object Definition.

Similar to the DataGroup Definition Editor, the DataGroup Catalog Editor is a stand-along application that can be launched from Windows or from with-in the BA application. This standalone Windows application that can be started two ways: from the Windows Start Menu via *Start*| *Bentley Building V8 XM*| *Building Tools*| *DataGroup*| *DataGroup Catalog Editor*, and it can also be started from within BA under the Architecture Menu| Schedules & Reports| Edit Catalog. The dialog box appears as in Figure 10.

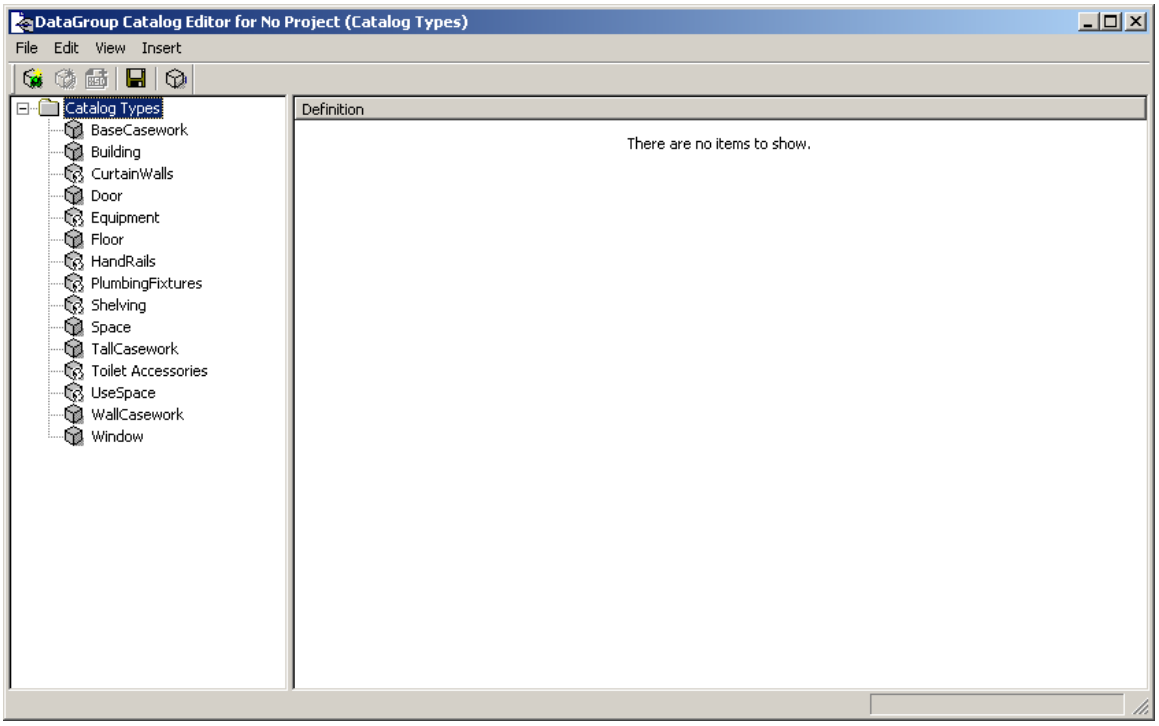

Figure 10 – DataGroup Catalog Editor.

The *DataGroup Catalog Editor* works in two modes. One mode is used to attach the *DataGroup Definition* to the *Catalog Type*, and the second mode is used to enter new or modify existing *Catalog Items* into the *Catalog*. The modes are controlled using the Show Catalog Items button, shown in Figure 11, 12 and 13. For the purpose of creating a custom object in BA, we will be using the editor in Show Catalog Off mode so we can create a new Catalog Type and attach data definitions to the catalog.

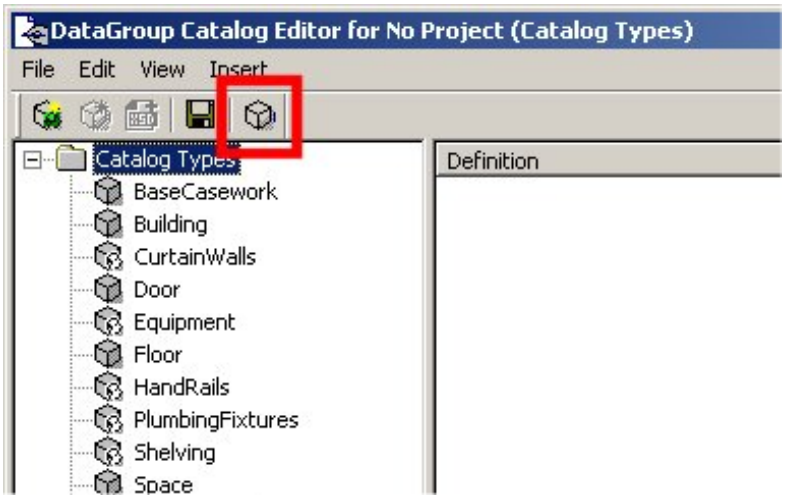

Figure 11 – DataGroup Catalog Editor, Show Catalog Items button.

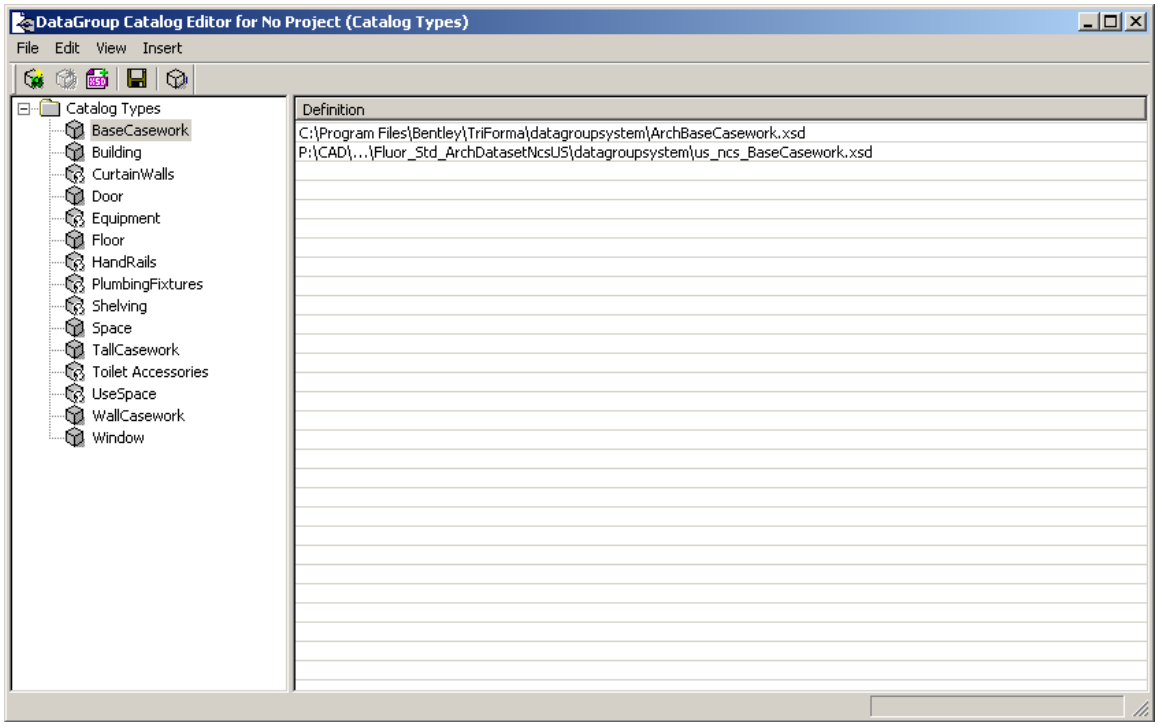

Figure 12 – DataGroup Catalog Editor, Show Catalog Items *OFF*.

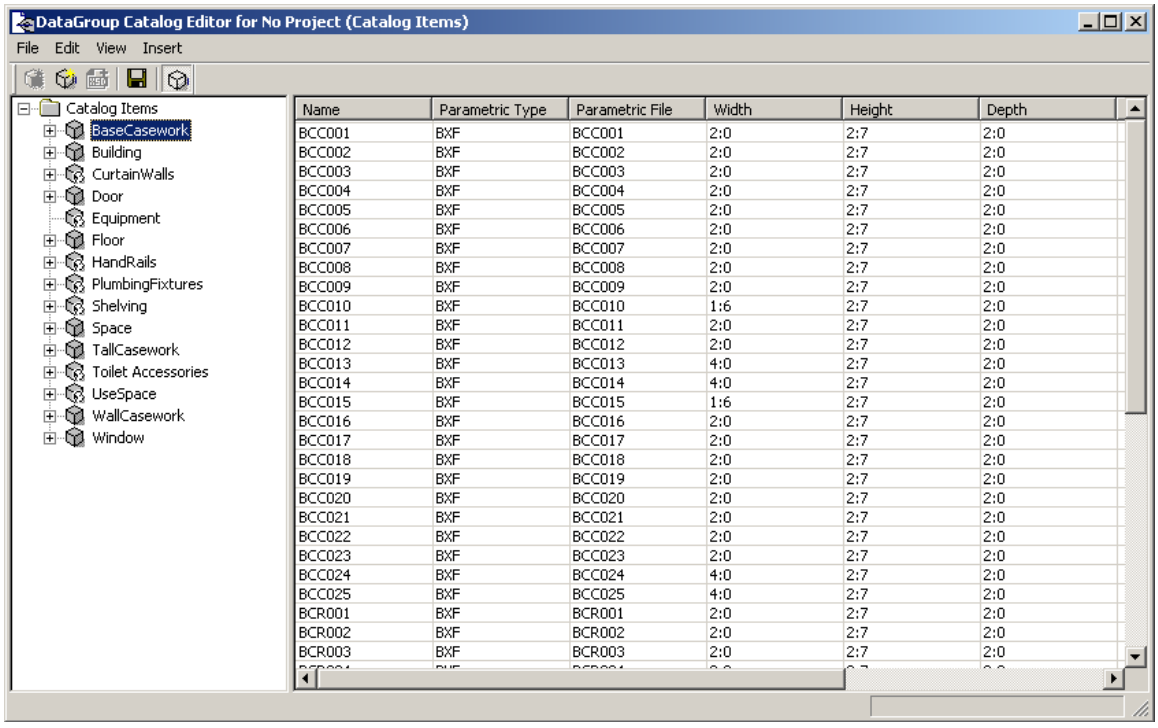

Figure 13 – DataGroup Catalog Editor, Show Catalog Items *ON*.

With the *Show Catalog Items* on, the *New Catalog Type* button (1) and the right-click pop-up menu *New Catalog Type* item are active (non-grayed). (See Figure 14) Selecting

either of these will create a Catalog Type. And the New Catalog Type dialog box is displayed. The dialog box is used for naming the type and giving it a filename. Enter the name for the Catalog in the Name text box (1) and then click on the Create a new DataGroup File button (2). (See Figure 15) The New File dialog box will then appear. (See Figure 16)

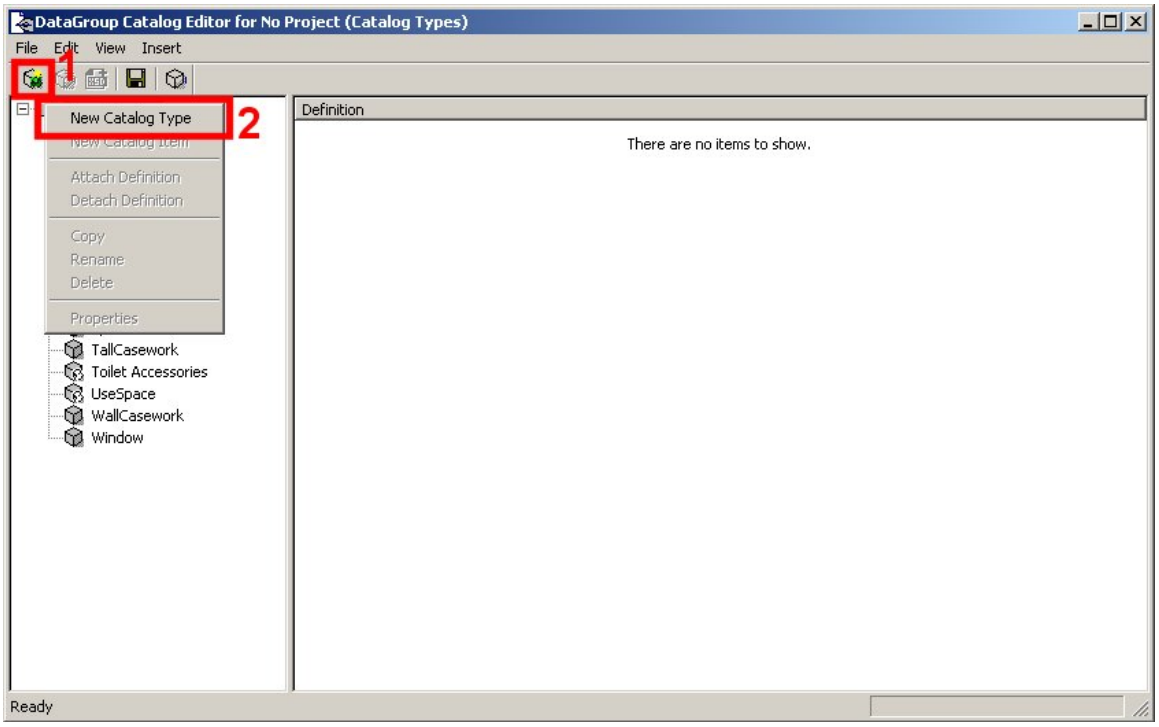

Figure 14 – Create New Catalog Type.

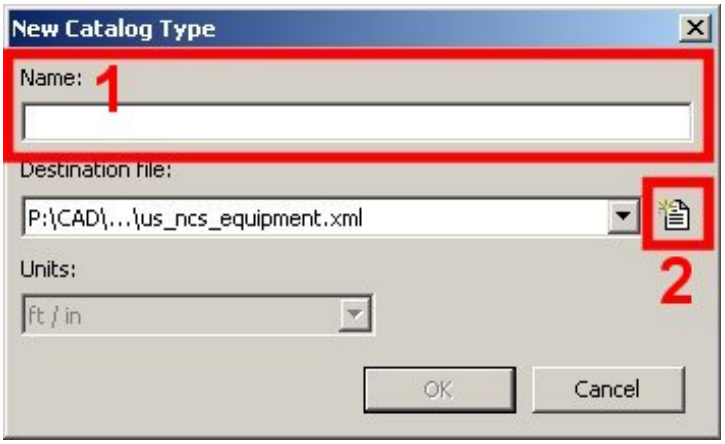

Figure 15 –New Catalog Type dialog box.

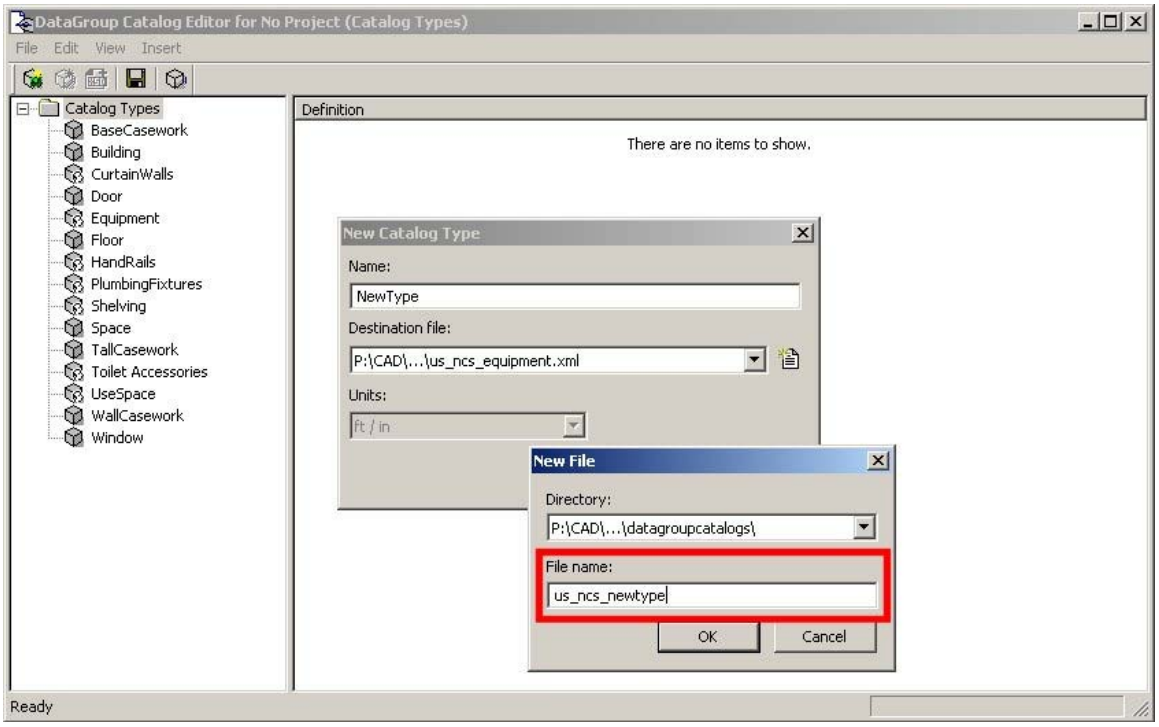

Figure 16 – New File dialog box.

Similar to the *DataGroup Definition*, *Catalog* files are created in the directory as specified in the New File dialog box. The default folder for catalogs is *\datagroupcatalogs*. It is not normally necessary to point to another location other than the default given. Once a name is entered and the OK button is clicked a definition file will be created in the given directory with the name entered and an extension of .xml.

It is worth taking a few moments to talk about file naming conventions. Again, Bentley has used the prefix "*us_ncs*", which stands for United States National CAD Standard. It is recommended that site specific prefix be used that reflects the company or project name. Here at Fluor we are using "*flr_*". Also note that in general, Bentley uses the convention of naming xsd files using no capital letters for catalog files (and xml files in general) (e.g *us_nc_**p**lumbing**f**ixtures* and not *us_nc_ **P**lumbing**F**ixtures*).

At this point in the process, we have an empty *Catalog* without a data structure. The next step is to attach a *DataGroup Definition* or *Definitions* to the *Catalog* so it will have a data structure. Right-click on the Catalog Type and select Attach Definition from the pop-up menu (1) or click on the Attach DataGroup Definition button (2). (See Figure 17)

Attaching the Definition to the Catalog Type is done in the Attach Definition dialog box. Using the drop-down box select the Definition to Attach created in the first part of this document. Next, using the drop-down box select the Destination File. Select the *catalogtypeexts.xml* file from the drop-down list (see the section titled Guide to BA Dataset Files below for additional information). Repeat the attachment process for each data definition required. This step marries the Definition with the Catalog. We now have completed the final step of creating a custom object. Now we can enter data into our Catalog to create individual Catalog Items that we can attach to individual instances of graphics in our design file.

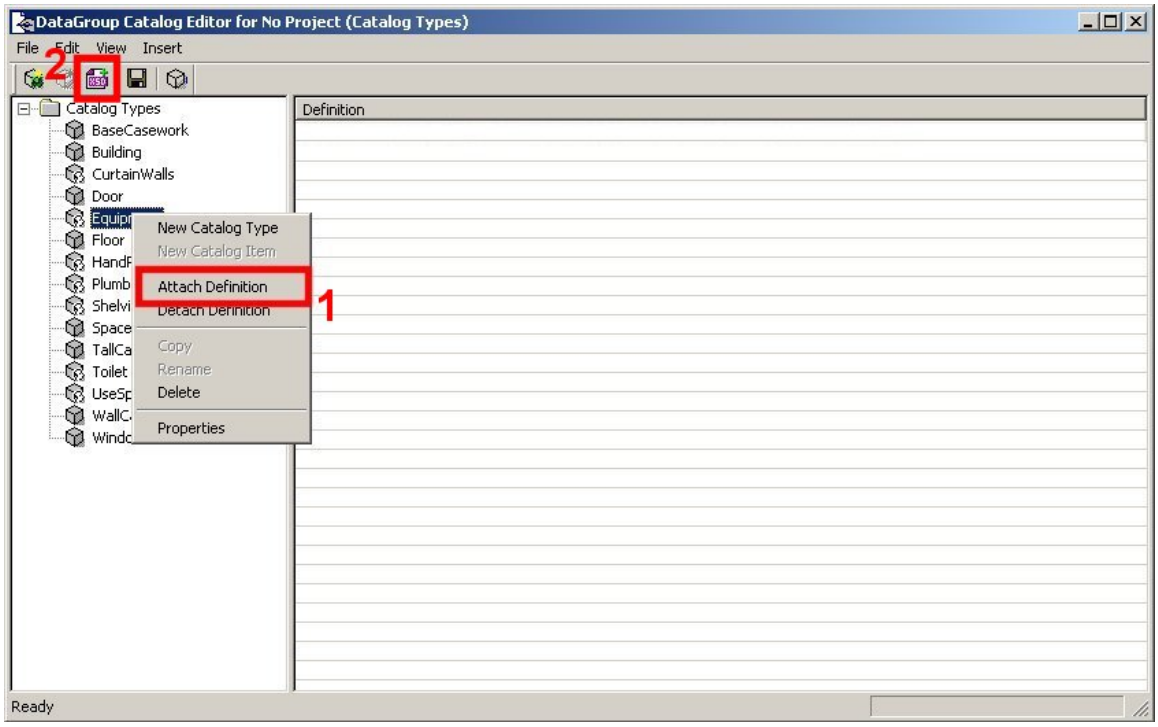

Figure 17 – Attach a DataGroup Definition to a Catalog Type.

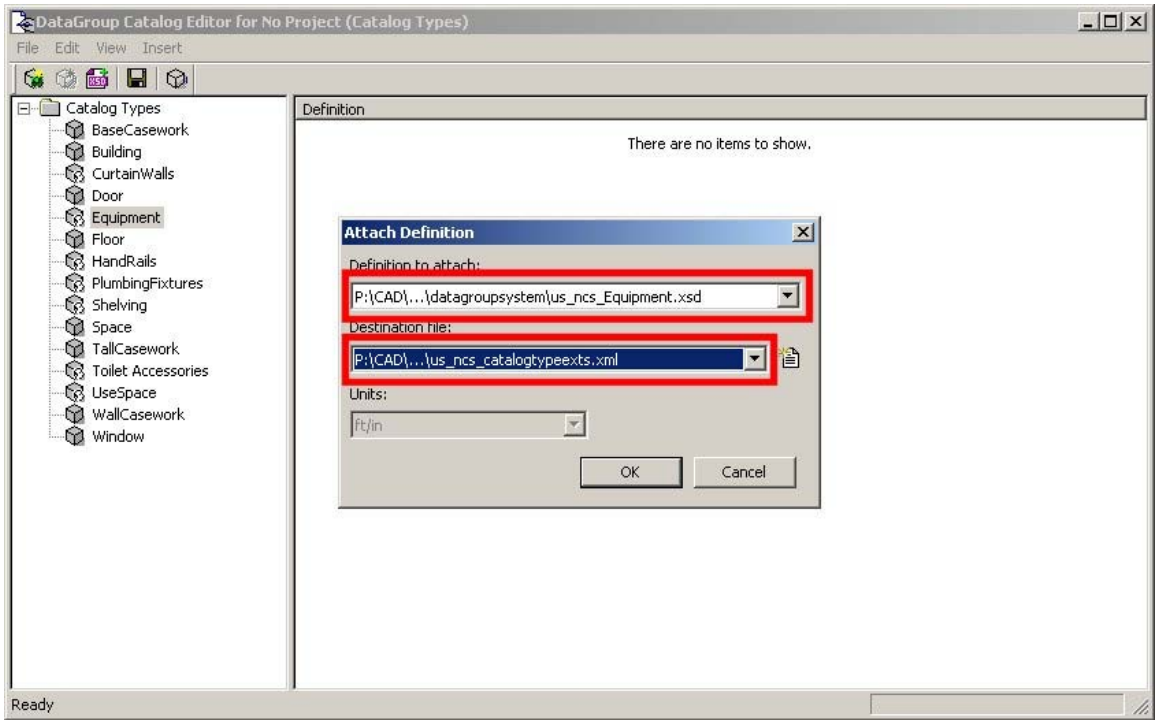

Figure 18 – Attach Definition dialog box.

Adding New Items to a Catalog Type

Set the DataGroup Catalog Editor to the Show Catalog Items *ON* mode of operation as depicted in Figure 13. A new Catalog Item is created by clicking on the New Catalog Item button (1) or from the Insert menu by selecting Catalog Item (2). (See Figure 19)

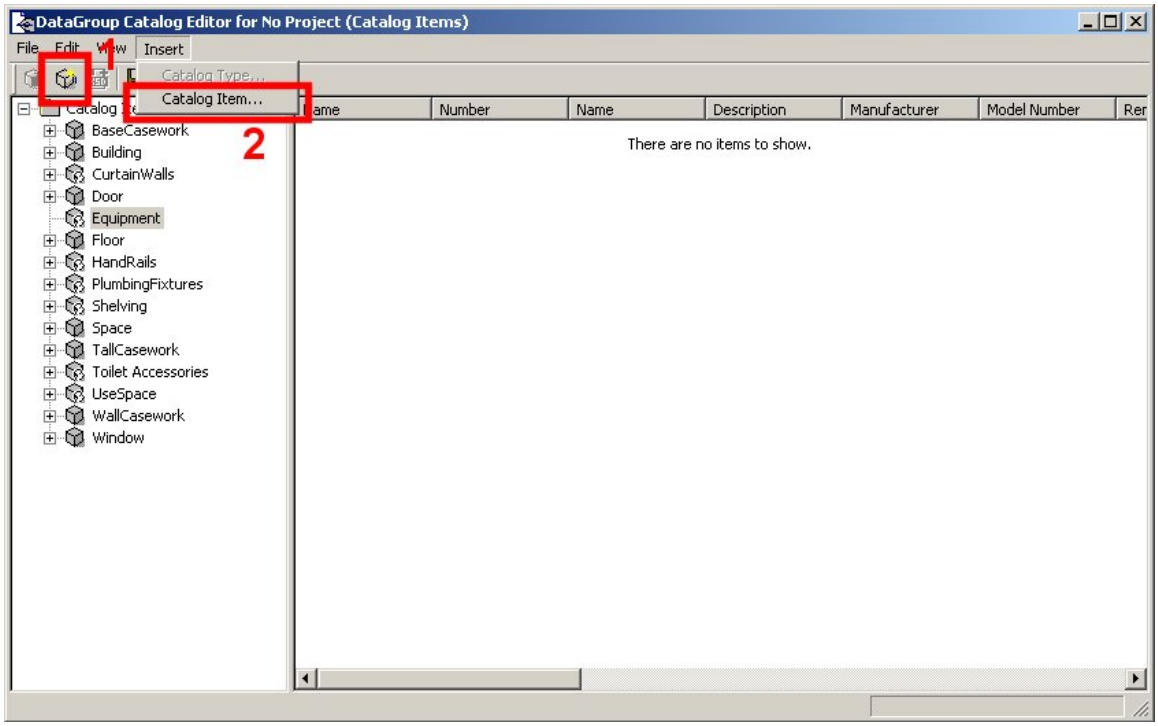

Figure 19 – Create a new Catalog Item.

This opens the New Catalog Item dialog box shown in Figure 20. Give a Name to the Item (1) and then select the Catalog Type from the drop-down list. (It does not seem to matter if the Catalog Type is hi-lighted (selected) in the tree on the left hand half of the application, you still need to select it in the drop-down list.)

Click the OK button and you will see the Item show up as hi-lighted (selected) in the tree on the left and the properties listed on the right half of the application as shown in Figure 21. Using the Property editor on the right half of the application enter the default values for the Catalog Item. Remember to save your work periodically.

To see a list of the Items in a particular Catalog Type, select the Catalog Type in the left hand side of the application as shown in Figure 13.

This completes the tutorial for creating a custom BIM object in Bentley Architecture. To attach a particular Catalog Item to a graphic element in the design file, simply use the *Add Instance* command in the Architecture| Schedules & Reports menu of BA. Note that you can add Instance data from the Catalog to items that already have data such as doors. So if you wish to add equipment data to a overhead-rolling door BA will allow the attachment. Once the Instances are added to the design file elements, you can run reports for schedules and estimating purposes as with any BA data elements.

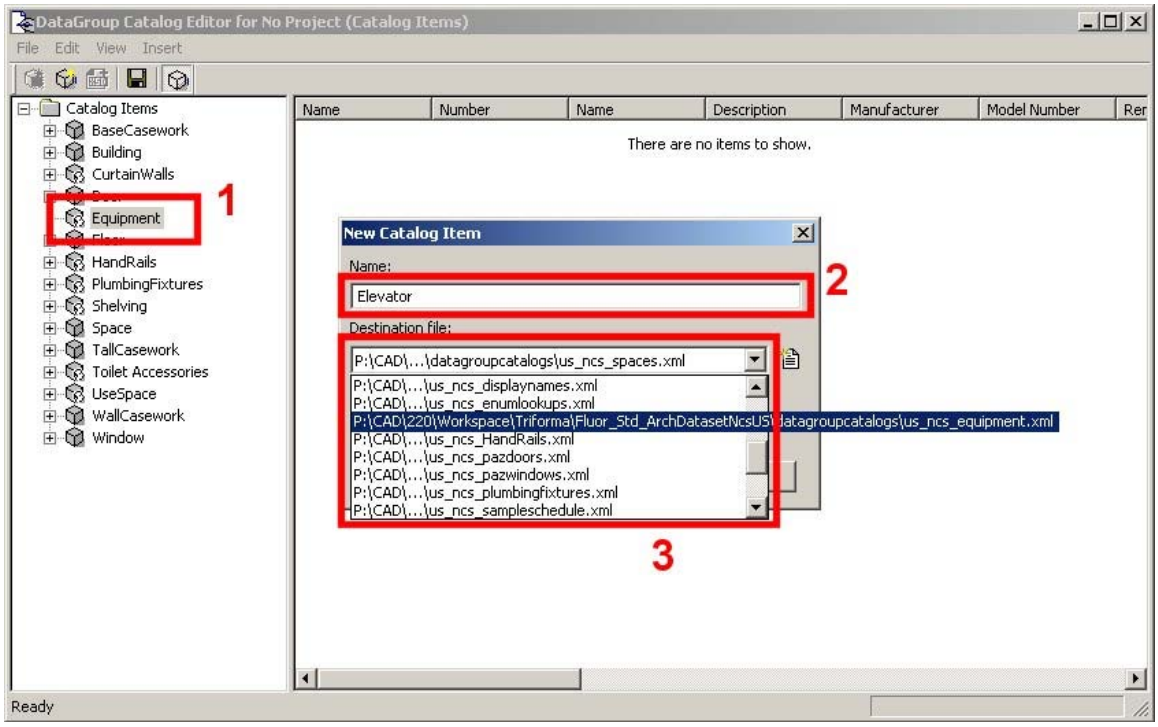

Figure 20 – New Catalog Item dialog box.

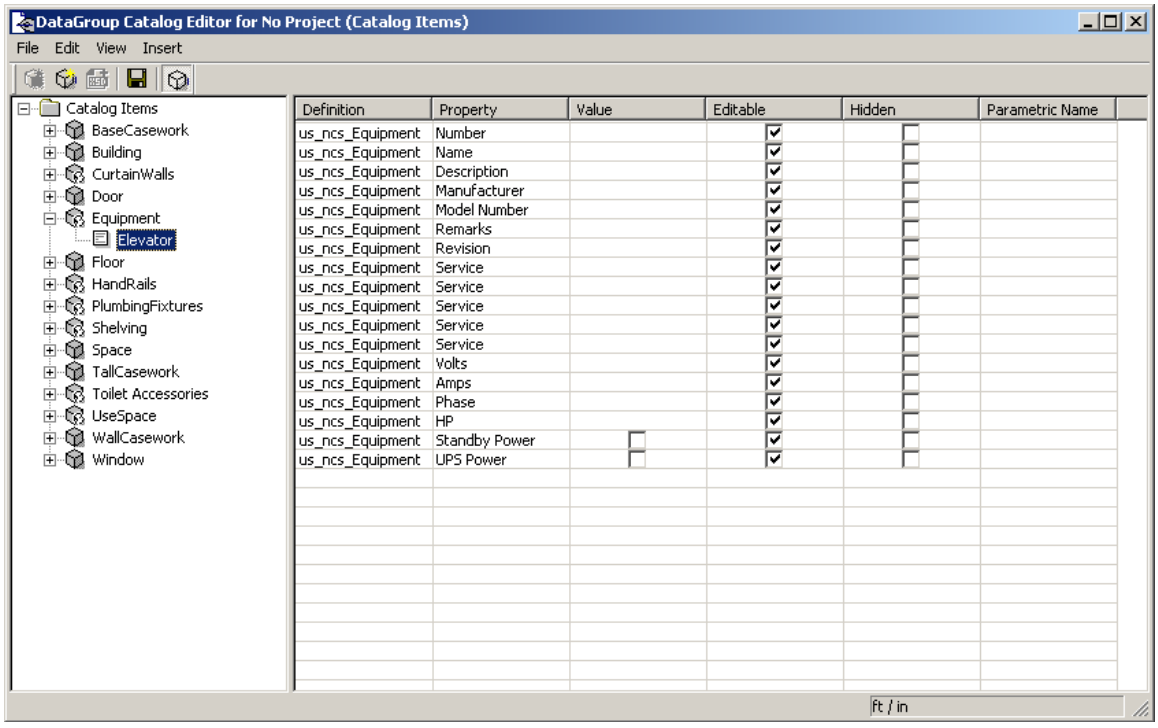

Figure 21 – Catalog Item Property Editor.

Guide to the BA Dataset Files

Bentley Architecture doesn't give a lot of guidance on how to use the files that make up the database or which ones should be used for what. This can be good and bad. Good because there is a lot of freedom in customizing the database. Bad, because without some level of guidance it is difficult for beginning users to understand how all of this should or could work.

When I first began to modify the database, adding definitions, catalogs, and items that we needed, I made a mess of things because I didn't have a sound working knowledge of what files contain what. The Data Definition Editor and especially the Catalog Editor would ask me for a filename and I had absolutely no idea what should be where or what conventions might apply. I followed several tutorials and guides and there seemed to be slight differences that just added to my consternation.

Bentley only offers remedial guidance. Without a good solid understanding of the basis and inner-workings of the BA database, I believe it is difficult to organize a company's database customizations. If they aren't organized then the customizations will be difficult to debug, it will be difficult to understand what you have, and most importantly it will be difficult or even impossible to do a simple port of all the database customizations to a new version of BA. Thankfully the system is so straightforward that with a little study someone can understand what Bentley intends.

The Bentley parser is very flexible. It can read stuff from all over a collection of files and as long as the syntax is correct it will gobble them down, organize them, and spit them out in an organized presentation in the editor interfaces. Things really don't have to be in the proper files as much as you think. At least when I made a mess of the database, by all appearances things looked fine in the editors. But when I got confused and began to wonder if I was using the proper files or naming conventions. When I went looking in the database files, I found out I had made a mess of things. Several things were in more than one file. That's when I sat down to study exactly where Bentley was putting what and how all of this worked in the background, under the hood og the definition and catalog editors.

The primary interfaces offered by BA for editing the Dataset files are the DataGroup Definition Editor and the DataGroup Catalog Editor. These are standalone programs that can be launched from inside Bentley Architecture or from the Start Menu. They operate on a set of files in the TriForma Dataset folder (as defined by the configuration variable TF_DATASETNAME). There are two sub-folders involved: *datagroupcatalogs* and *datagroupsystem*. The DataGroup Definition Editor works with the files in the *datagroupsystem* folder. And as expected, the DataGroup Catalog Editor works with the files in the *datagroupcatalogs* folder. However useful and fit for purpose these programs are, they can be difficult to use and understand without understanding the foundation of XML files on which they rely and which are the foundation for the TriForma database.

What's XML?

The files are ASCII (text) files and may be edited with a simple text editor. They are in a format commonly known as Extensible Markup Language (XML). XML is an open-format, general purpose specification for creating custom-designed languages. There is a lot of information on the web concerning the general XML language specification and so that won't be our focus here. And, while there is a general specification that allows for the

parsing of the language, it is intended to be further defined into a custom-designed, fit for purpose language. Bentley has done just that.

A lot of folks will say, “Ah, you don’t need to know all of the XML stuff. That’s for programmers and advanced users. If you go into those files you’ll mess things up.” However, I think it is worth being able to read the stuff even if you don’t go in and edit it at the text level. It is also worth the time to point out a few basics of XML. The first line of an XML file declares the specification, or version level of the XML language. It is called the XML Declaration, and it looks something like this:

```
<?xml version="1.0" encoding="Windows-1252"?>
```

There is no reason to change this line in any of the Bentley XML files. If you decide to create a XML file from scratch, you’ll need to include this or the Bentley parser won’t even read past the first line in the file. And it is worth saying at this point that the program that reads the XML file is referred to as a parser. A parser is just a fancy term for a program that is used to read a computer language. Do take notice about one thing in the construction of this line, it begins with a “<” and ends with a “>”. These brackets always surround a subject line in an XML file. A subject line in XML is termed a *tag*.

After the first tag the file consists of a start tag and an end tag that frames information. It’s kind of like saying I’m beginning a new topic, and then after the subject has been discussed you say I’m ending my subject. For multiple lines, it looks something like this:

```
<CatalogItems>
   ... some sort of information ...
</CatalogItems>
```

For single lines, it looks something like this:

```
<CatalogItem type="flr_LabCasework" name="Default Lab Casework"/>
```

Notice the already familiar begin < and end > control characters are there in both cases. Also note that there is a new control character, the / is used. The / is a signal to the parser that the end has come. A / means *end*.

So, multiple lines are framed by a tag and end in the same tag, only with a /. Multiple lines are sometimes called a *logical component*. A logical component of a document which begins with a start tag and ends with an end tag. The tag can be any label you might want to identify as an entity.

```
<EntityTag>
   ... information that defines the entity ...
</EntityTag>
```

Logical components can be nested inside each to form sub-components.

```
<EntityGroup>
   <EntityTag1>
      ... information that defines the entity ...
   </EntityTag1>
   <EntityTag2>
      ... information that defines the entity ...
   </EntityTag2>
</EntityGroup>
```

A single line, or tag, is termed an *attribute*, and they are placed inside, or framed, by a logical component. One or more attribute tags contain the information that defines the logical component.

```
<EntityGroup>
   <EntityTag1>
      <EntityItem name="Default"/>
      <EntityItem name="Option 1"/>
      <EntityItem name="Option 2"/>
   </EntityTag1>
   <EntityTag2>
      <EntityItem name="Default"/>
      <EntityItem name="Option 1"/>
      <EntityItem name="Option 2"/>
   </EntityTag2>
</EntityGroup>
```

The format for an attribute tag is simple. Begin with a <, followed by a name for the tag, then one or more equality statements, and finally a />. Equality statement is programmer talk for a statement in the form of X=Y – something equals something.

And that's about it. The format of an XML file is relatively simple thing. But what makes it so useful and rich is that it can be used to represent a multitude of ideas, data, and documents. As an example of its power, the HTML language use by Internet browsers is a special case of XML. And, while HTML can be a very complex language, as implemented by Bentley for a special purpose langue used in creating their Datasets, it is relatively simple and straight-forward.

I guess there is only one last thing to say about XML files. And while it won't affect your understanding of Bentley Datasets, it is one of the other foundations concepts of XML. Living in a computerized, web-driven world, you may have heard of the term *namespace* and wondered, "What the heck is a namespace?"

Earlier I said that XML is used as a general purpose specification for creating custom-designed languages. Well, this might also be said that XML is used as a general language for creating specific languages that represent special concepts. In general, a *namespace* is defined as a general container that provides context for specific items. That's exactly what XML is! So, when XML is cast to a solve a specific need, it provides context. A specific implementation of XML to solve a particular need is a *namespace*.

Bentley DataGroups

All databases are formed of two primary parts: the data definition (sometimes called the schema), and the data. The data definition is the framework for the data. Much like a skeleton, it supports the data in an organized manner. So, based on the traditional database concept, Bentley has split their Dataset/DataGroup system into two functions.

The two programs they use to operate on the TriForma database and the two corresponding directory folders reflect this.

First, let's look at the database definition, or schema. This special purpose XML file has the extension .xsd. It uses a specific implementation of XML that is used to represent a database schema. The first line in the file, as with all XML, establishes that the file uses the XML specification. The second line in the file establishes that the file uses a namespace used to define database schemas. The last line in the file is an end tag for the schema.

```
<?xml version="1.0" encoding="Windows-1252"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

 ... definition of the database schema ...

</xs:schema>
```

While it is possible to include data definitions for more than one entity in an XML Schema file, Bentley chooses to dedicate a file to each entity type. So, there are individual files that define the data definition for Buildings, Walls, Floors, Door, etc. If you are using the Bentley U.S. National CAD Standard dataset, then the as-delivered files begin with "ncs_us_" followed by the name of the entity. It is recommended that another prefix be used to differentiate any new entities created. Here at Fluor, I am using "flr_".

As an example, let's look at the schema file for Buildings in Bentley Architecture. This file resides in the *\datagroupsystem* folder and is named ncs_us_Building.xsd. (Notice they use the naming convention where a capital-B is used in the word Building, and this convention was discussed earlier.)

```
<?xml version="1.0" encoding="Windows-1252"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:simpleType name="bVolumeUnitsType">
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:simpleType name="bAreaUnitsType">
      <xs:restriction base="xs:string"/>
   </xs:simpleType>
   <xs:element name="us_ncs_Building">
      <xs:complexType>
         <xs:attribute name="GrossArea" type="bAreaUnitsType" use="optional"/>
         <xs:attribute name="RawArea" type="bAreaUnitsType" use="optional"/>
         <xs:attribute name="NetArea" type="bAreaUnitsType" use="optional"/>
         <xs:attribute name="GrossVolume" type="bVolumeUnitsType" use="optional"/>
         <xs:attribute name="NetVolume" type="bVolumeUnitsType" use="optional"/>
         <xs:attribute name="SiteCoverage" type="bAreaUnitsType" use="optional"/>
         <xs:attribute name="Remarks" type="xs:string" use="optional"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

This schema begins by defining two types of data that will be used through-out the definition of the data that follows, the *bVolumeUnitsType* and the *bAreaUnitsType*. This

is typical. Some of the first lines in a given schema file define the types of special data that will be used in the individual attribute definitions.

The supported Bentley data types are (see Figure 22): String, Integer, Decimal, Booolean, Time, URL, Date, Date and Time, List (Enumerated list), Working Units, Area Units, and Volume Units. Four of these appear in the XML file as special Bentley types: *bWorkingUnitsType*, *bVolumeUnitsType*, *bAreaUnitsType*, *bEnumLookupType*. (note that their names all begin with the signature lower-case letter "b".) When these types are used by an attribute in the data definition, they must be declared at the beginning of the file similarly to the ones in ncs_us_Building.xsd.

After the data definitions, the name of the data definition is given in the *element_name* tag: `<xs:element name="us_ncs_Building">` and the end of the definition is declared with the tag `</xs:element>`. Between the two logical component tags the individual attributes of the data definition are given. The attribute tag contains information concerning the attribute name [name=], type of data [type=], and if the data is required or not [use=optional | required].

Figure 22 shows what the Building definition file looks like when it is viewed using the DataGroup Definition Editor.

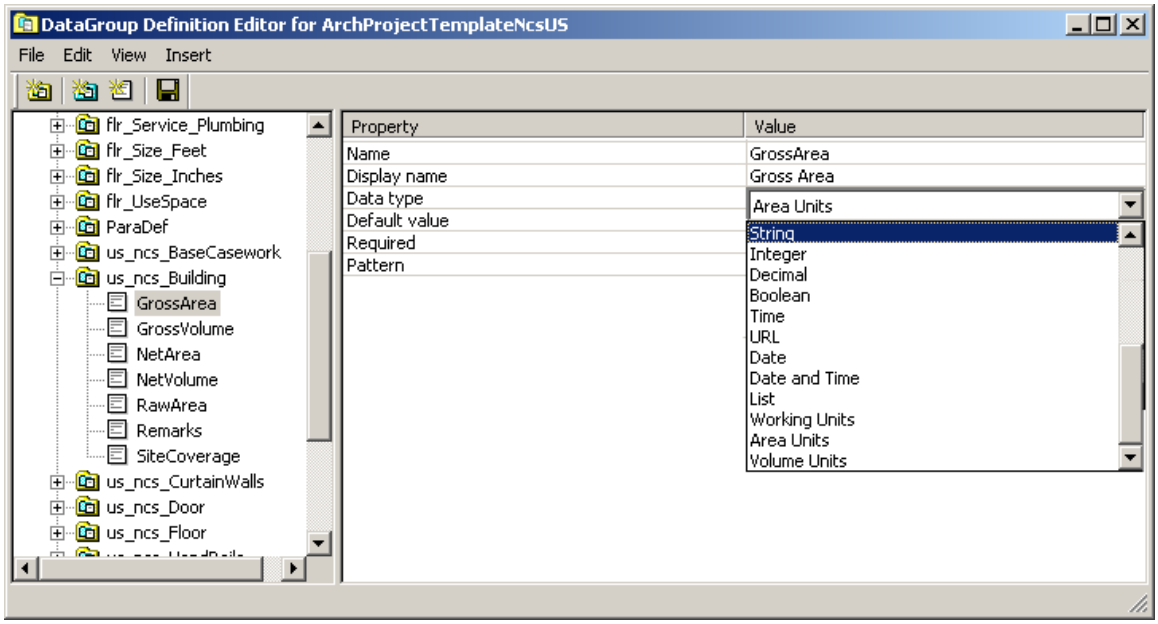

Figure 22 – DataGroup Definition Editor, Data Types.

Special Purpose DataGroup Files

Besides the individual files used to define the data definition for the various entities, there are several special files located in the*datagroupsystem* folder: *us_ncs_enumlookups.xml*, *us_ncs_displaynames.xml*, *us_ncs_catalogtypeexts.xml*, and *MRULookups.xml*. Each of these files fulfills a unique need.

Special Purpose DataGroup Files – The Enumerated List

The *us_ncs_enumlookups.xml* file contains the dedicated values for lists that are used in conjunction with attributes that use the *bEnumLookupType* datatype. The Enum part of the name represents *Enumeration*. Enumeration means to specify one thing after another, as in a list of values. Here it is a list of values that make up the selections that uses may select for a given property.

The <*DataGroupSystem*> tag followed by <*EnumLookups*> tag begins the logical component that defines the groups of lists. Each list is sub-defined in a logical sub-component. The header of this logical component is structured similar to an attribute. It begins with the tag <*EnumLookup* and is followed by 1) *definition*=, which is used to give the data definition/filename to which the list relates; followed by 2) *property*=, which points to the specific property of the definition; and followed by 3) *extendable*=, which tells the system if a user may (true) or may not (false) add to the list. Nested inside the <*EnumLookup* component are the individual <Enum attribute tags that define the list entries.

The *us_ncs_enumlookups.xml* listed below is the as-delivered version. Figure 23 shows the *MountType* enumerated list as it appears in the DataGroup Definition Editor.

```
<?xml version="1.0" encoding="Windows-1252"?>
<DataGroupSystem>
   <Version major="1" minor="0"/>
   <EnumLookups>
      <EnumLookup definition="us_ncs_ToiletAccessories"
property="us_ncs_ToiletAccessories/Type/@MountType" extendable="true">
        <Enum value="Surface Mounted"/>
        <Enum value="Recessed"/>
        <Enum value="Free Standing"/>
        <Enum value="Suspended"/>
     </EnumLookup>
     <EnumLookup definition="us_ncs_ToiletAccessories"
property="us_ncs_ToiletAccessories/Extra/@MountBy" extendable="true">
        <Enum value="Top AFF"/>
        <Enum value="Bottom AFF"/>
     </EnumLookup>
   </EnumLookups>
</DataGroupSystem>
```

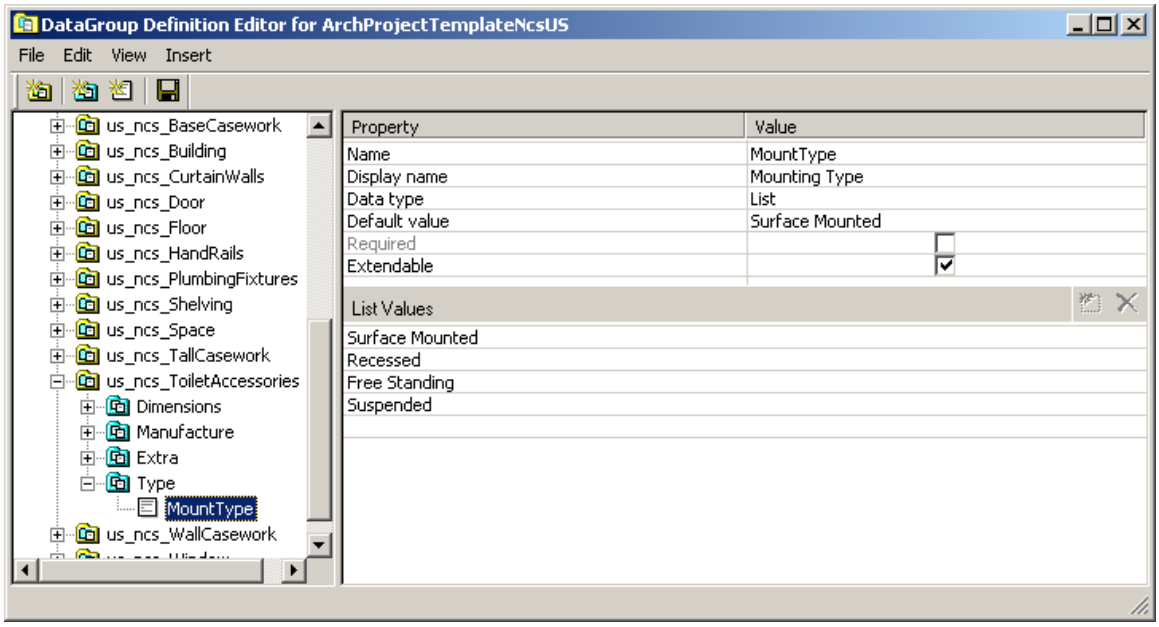

Figure 23 – DataGroup Definition Editor, List Values.

Due to restriction in the way enumerated lists were implemented here, even if a list is the same from one property to another, the list must be repeated for each property. In the example below Service 1, Service 2, and Service 3 all share identical enumerated lists, but each must be repeated in the enumeration file. This is an unfortunate circumstance, that perhaps Bentley will fix at some time in the future.

```
<EnumLookup definition="flr_Service_Lab" property="flr_Service_Lab/@Service1"
extendable="true">
   <Enum value="Water, Cold Potable (CPW)"/>
   <Enum value="Water, Hot Potable (HPW)"/>
   <Enum value="Floor Drain, Open Hub"/>
   <Enum value="Floor Drain, Sealed Hub"/>
   <Enum value="Floor Drain, Trench"/>
   <Enum value="None"/>
</EnumLookup>
<EnumLookup definition="flr_Service_Lab" property="flr_Service_Lab/@Service2"
extendable="true">
   <Enum value="Water, Cold Potable (CPW)"/>
   <Enum value="Water, Hot Potable (HPW)"/>
   <Enum value="Floor Drain, Open Hub"/>
   <Enum value="Floor Drain, Sealed Hub"/>
   <Enum value="Floor Drain, Trench"/>
   <Enum value="None"/>
</EnumLookup>
EnumLookup definition="flr_Service_Lab" property="flr_Service_Lab/@Service3"
extendable="true">
   <Enum value="Water, Cold Potable (CPW)"/>
   <Enum value="Water, Hot Potable (HPW)"/>
   <Enum value="Floor Drain, Open Hub"/>
   <Enum value="Floor Drain, Sealed Hub"/>
   <Enum value="Floor Drain, Trench"/>
   <Enum value="None"/>
</EnumLookup>
```

Special Purpose DataGroup Files – Display Names

The *us_ncs_displaynames.xml* file contains the names of the properties that are used for display in the DataGroup Catalog Editor and in schedules. The format for the Display Names file is relatively simple. An example of a portion of a typical file is given below. The logical component tag is a simple <*DisplayNames*> and each attribute in the group is an individual Display Name.

```
<?xml version="1.0" encoding="Windows-1252"?>
<DataGroupSystem>
   <Version major="1" minor="0"/>
   <DisplayNames>
      <DisplayName name="ArchSpace/@label" displayName="Label"/>
      <DisplayName name="ArchSpace/@number" displayName="Number"/>

      ...

      <DisplayName name="us_ncs_Door/Extra/@Roomname" displayName="Room Name"/>
      <DisplayName name="us_ncs_Door/@Roomname" displayName="Room Name"/>
      ...
   </DisplayNames>
</DataGroupSystem>
```

Special Purpose DataGroup Files – Catalog Type Extensions

The *us_ncs_catalogtypeexts.xml* file contains the logical link between the DataGroup Definition and the DataGroup Catalog files. For the most part, for the files that reside in the *datagroupsystem* folder the interface for creating and editing them is the DataGroup Definition Editor, and the ones in the *datagroupcatalogs* folder are created and edited using the DataGroup Catalog Editor. This file is different in that it resides in the *datagroupsystem* folder, but the interface is through the Catalog Editor. The Catalog Editor has two modes, making it two different interfaces in some respects. When the *Show Catalog Items* is set to off, then the mode for the interface is to link the catalogs with the definitions. In this mode the destination file in the pop-up dialog box determines where the <*/CatalogTypeExtensions*> tag is written.

There is more than one philosophy about how to organize the information in the XML files. Bentley's flexible parser will find the data definition link as long as it is contained in an XML file along the DataGroup Path (DG_PATH configuration setting). And while there is more than way to look at how this is done, a single philosophy should be chosen and used consistently.

Bentley has chosen to always put them in *us_ncs_catalogtypeexts.xml* file, and you may choose to follow their example. However for ease in migration (see the discussion below for more on this), I recommend that you segregate your company or site data definitions and catalogs from those of Bentley. This can be done in one of two ways.

Here at Fluor, we have chosen to use a *flr_ catalogtypeexts.xml* file. It mirrors the Bentley provided *ncs_us_catalogtypeexts.xml* file. So when we link our definitions with the catalogs, we specify our file in the destination file box of the pop-up dialog box.

Another method, just as valid in my mind, is to specify the XML file that contains the individual catalog items as the destination file (more on this later). Each is a valid philosophy. However as stated previously consistency counts, choose one way or another and stick to the method.

When I first tried to create custom site specific data definitions and catalogs, I followed several examples by others. Without realizing it that I had copied examples of both philosophies. Soon I had <*CatalogTypeExtensions*> tags in both individual catalog item XML files as well as in the Bentley provided *us_ncs_catalogtypeexts.xml* file. This made quite a mess of things.

Now that we have discussed the use of the *catalogtypeexts.xml* file, we can look at its internal XML structure and of the <*CatalogTypeExtensions*> logical component. Inside of the overall */CatalogTypeExtensions*> tag, there are nested <*CatalogTypeExtension* logical component tags. As part of the <*CatalogTypeExtension* logical component tag, the name of the Catalog Type is given in double-quotes after the token expression *type*=.

Nested inside of this are the individual attribute tags for the data definition(s), the <*InstanceDataDefinition* tag. The format for the <*InstanceDataDefinition* attribute tag is relatively simple. Following the tag label in the syntax is *defType*= and this is always set to the value of *"User"*. Next is *definition*=, which is set to the name of the XML file that defines the data structure (schema) that is to be used for the catalog items. More than one definition can be attached to create the definition of the data for a catalog item.

```
<?xml version="1.0" encoding="Windows-1252"?>
<DataGroupSystem>
   <Version major="1" minor="0"/>
   <CustomCatalogTypes>
      <CustomCatalogType name="Toilet Accessories"/>
      <CustomCatalogType name="Shelving"/>
      <CustomCatalogType name="PlumbingFixtures"/>
   </CustomCatalogTypes>
   <CatalogTypeExtensions>
      <CatalogTypeExtension type="BaseCasework">
         <InstanceDataDefinition defType="USER" definition="us_ncs_BaseCasework"/>
      </CatalogTypeExtension>
      <CatalogTypeExtension type="Door">
         <InstanceDataDefinition defType="USER" definition="us_ncs_Door"/>

      ...

      <CatalogTypeExtension type="PlumbingFixtures">
         <InstanceDataDefinition defType="USER" definition="ParaDef"/>
      </CatalogTypeExtension>
      <CatalogTypeExtension type="PlumbingFixtures">
         <InstanceDataDefinition defType="USER"
definition="us_ncs_PlumbingFixtures"/>
      </CatalogTypeExtension>
      <CatalogTypeExtension type="Shelving">
         <InstanceDataDefinition defType="USER" definition="ParaDef"/>
         <InstanceDataDefinition defType="USER" definition="us_ncs_Shelving"/>
      </CatalogTypeExtension>
   </CatalogTypeExtensions>
</DataGroupSystem>
```

When multiple definitions are attached to a Catalog Type, the DataGroup Catalog Editor creates a <*CatalogTypeExtension* logical component for each of the attachments. It only writes one <*InstanceDataDefinition* inside of a <*CatalogTypeExtension*. In the following instance two data definitions, us_ncs_ToiletAccessories and ParaDef, are being linked to the Catalog Type *Toilet Accessories*.

```
<CatalogTypeExtension type="Toilet Accessories">
   <InstanceDataDefinition defType="USER" definition="us_ncs_ToiletAccessories"/>
</CatalogTypeExtension>
<CatalogTypeExtension type="Toilet Accessories">
   <InstanceDataDefinition defType="USER" definition="ParaDef"/>
</CatalogTypeExtension>
```

However, once again, the Bentley parser is flexible, and it will also read more than one *<InstanceDataDefinition* nested inside of a *<CatalogTypeExtension*. The parser would read the following code the same way it read the previous code.

```
<CatalogTypeExtension type="Toilet Accessories">
   <InstanceDataDefinition defType="USER" definition="ParaDef"/>
   <InstanceDataDefinition defType="USER" definition="us_ncs_ToiletAccessories"/>
</CatalogTypeExtension>
```

Since the parser reads both identically and the latter reads more clearly, it is suggested that you follow this convention if you choose to manually editing a Catalog Type Extension file.

At this point it is worth mentioning the use of the *ParaDef* data definition. Linking the *ParaDef* to a catalog type item allows TriForma to associate catalog items with cells – both parametric and non- parametric. *ParaDef* has two properties: *Type*, and *Filename*. Type can be CEL, BXC, or PAZ. PAZ cells are created in PC Studio. BXC, TriForma *Compound Cells*, created using the *Compound Cell Manager*. CEL are traditional MicroStation cells. For CEL and BXC parametric items, the *File Name* property is the same as the name of the cell. For PAZ parametric items, the *File Name* is the same as the item name without the extension.

(It is important to note that graphic item files be located on the ATFDIR_CELL directory path, and bear the same name and parametric definition used in the setup procedures. Check MicroStation Triforma / Bentley Architecture configuration to insure that ATFDIR_CELL points to the correct location. See MicroStation Help for additional documentation)

Special Purpose DataGroup Files – MRU Lookups

The *MRULookups.xml* file file contains a list of the Most Recently Used catalog item names. The name is a bit of a misnomer. You might expect, based on the name, that this is list is automatically generated the way a recent file list might be generated by a program for the file open operation. However this is a list generated by Bentley as a starting point and users can add to it. It is offered for the convenience of users so that they can have common key-in entries at their fingertips.

MRU Lookups are managed through a third, special interface not yet discussed. It is the MRU Manager. Use the key-in “DG MRU MANAGER” to initiate the manager. MRU Lookups are in many ways like the Enumerated Lookups, with the exception that there is no option to make them “extendable” in the attribute tag because it is not necessary.

```
<?xml version="1.0" encoding="Windows-1252"?>
<DataGroupSystem>
   <Version major="1" minor="0"/>
   <Units master="" sub=""/>
   <DataGroupSystemPrefs>
      <MRULookups>
         <MRULookup item="us_ncs_Space/@height">
            <MRULookupItem name="8"/>
            <MRULookupItem name="9"/>
            <MRULookupItem name="10"/>
            <MRULookupItem name="11"/>
            <MRULookupItem name="12"/>
            <MRULookupItem name="14"/>
         </MRULookup>

            ...

         <MRULookup item="us_ncs_Space/NorthFinish/@base">
            <MRULookupItem name="Rubber"/>
            <MRULookupItem name="Ceramic Tile"/>
            <MRULookupItem name="Cove Vinyl"/>
            <MRULookupItem name="Marble"/>
            <MRULookupItem name="Quarry Tile"/>
            <MRULookupItem name="Radial Rubber Tile"/>
            <MRULookupItem name="Slate"/>
            <MRULookupItem name="Terrazzo"/>
            <MRULookupItem name="Wood"/>
         </MRULookup>
      </MRULookups>
   </DataGroupSystemPrefs>
</DataGroupSystem>
```

Migration of Special Purpose DataGroup Files

All of the file concepts I'm discussing throughout this paper can be accomplished using the DataGroup Definition Editor or Catalog Editor. I'm explaining the background XML so that you can use those interfaces with more intelligence and insight. However, I'm going to make a suggestion here that varies from what those two interfaces can do. When using the Definition Editor, the enumeration feature does not allow you to split the lists into those that are defined by Bentley and those that are defined by a site.

Let me say I am a strong proponent of keeping the Bentley delivered data in one set of files and the local, site defined data in totally separate files. It eases migration to future releases. Simply copy your definition and catalog files into the upgraded directories without fear of overlap. In this single instance, Bentley was unable to provide the ability to keep these separate. When you create a property as a list and define the values, they all go into the same file – *us_ncs_enumlookups.xml*. It is the same for the *us_ncs_displaynames.xml us_ncs_catalogtypeexts.xml*, and *MRULookups.xml*. I suggest as you take any site defined enumerations (those not defined by Bentley), and you copy them into another .XML file dedicated to those.

Again, here at Fluor, I am using "flr_" as a prefix to database file names that we create. So, we have a file named *flr_enumlokups.xml* that follows the strict format of the Bentley enumeration file, but it has only our Fluor enumerations in it. If the XML file format and tag conventions are followed, and it is located in the *\datagroupsystem*

folder, the Bentley parser will find it and read it. The same for the Display Names and Catalog Type Extensions.

I should also say that it is a good idea for most users to make database changes through the editor interface provided by Bentley and not by using a text editor. If you are an advanced user, using a text editor can provide certain advantages. However, if you make a mistake, you can disable portions of your database. In this one case, I feel it is a good idea to use a text editor to make data definition changes. Even if you don't separate them, and leave all the enumerations, display names, and catalog extensions each in individual Bentley delivered files, when it comes time to migrate to a new version you can use this knowledge to copy your site's data definitions to the new version's files.

Catalog Files

Up to this point we have been discussion the files located in the *datagroupsystem* folder. Now we'll move on to the files in the *datagroupcatalogs* folder. This is the other half of a database that was mentioned earlier. While most of the former files had an extension of xds and only a relative few had an xml extension, the files in this folder all have a xls extension. As far as the language used for any of these files, it is all XMS regardless of the extension. Its just that the two extensions help us differentiate between data definitions and the actual catalogs of data items.

As stated earlier, the DataGroup Catalog Editor is used as the primary interface for the files created and edited in the *datagroupcatalogs* folder. To be more precise and to expound on something already said, the DataGroup Catalog Editor portrays two modes of operation and it is in the Show Catalog Items mode that the editor works on files in this directory.

Each file in the directory is a catalog of items, their property values, and parametric names where appropriate. Not all of the properties from the linked DataGroup Definitions are represented. Only those with default or pre-assigned values or parametric names exist as attribute tags. The following is a typical catalog file.

```
<?xml version="1.0" encoding="Windows-1252"?>
<DataGroupSystem>
   <Version major="1" minor="0"/>
   <Units master="'" sub="&quot;"/>
   <CatalogItems>
      <CatalogItem type="Space" name="General">
         <Defaults>
            <Property definition="ArchSpace" name="ArchSpace/@label"
value="General"/>
            <Property definition="ArchSpace" name="ArchSpace/Area/@program"
value="100"/>
            <Property definition="ArchSpace" name="ArchSpace/@height" value="9:0"/>
         </Defaults>
      </CatalogItem>
      <CatalogItem type="Space" name="Administration">
         <Defaults>
            <Property definition="ArchSpace" name="ArchSpace/@label"
value="Administration"/>
            <Property definition="ArchSpace" name="ArchSpace/Area/@program"
value="200"/>
            <Property definition="ArchSpace" name="ArchSpace/@height" value="9:0"/>
         </Defaults>
      </CatalogItem>

      ...

      <CatalogItem type="Space" name="Corridor">
         <Defaults>
            <Property definition="ArchSpace" name="ArchSpace/@label"
value="CORRIDOR"/>
         </Defaults>
      </CatalogItem>
      <CatalogItem type="Space" name="Stair">
         <Defaults>
            <Property definition="ArchSpace" name="ArchSpace/@label" value="STAIR"/>
         </Defaults>
      </CatalogItem>
   </CatalogItems>
</DataGroupSystem>
```

By now the format is well recognized. Beginning with the tag <*DataGroupSystem*>, followed by <*CatalogItems*>, then <*Defaults*> and finally the individual items under the attribute tag <*CatalogItem*. The catalog item tag contains two tokens, the *Type*= and *Name*=. Since a single XML file contains a single catalog that the *Type* would be the same for all catalog items in a given *Catalog Type* file. (Although this does not have to be true and the parser would read catalog items from various catalogs and still sort out the resulting DataGroup Catalog Editor display in an orderly fashion. This is not recommended for obvious reasons of organization and consistency.)

When using the DataGroup Catalog editor to create a Catalog Item, the Destination File should be the XML file for that distinct Catalog Type. So, for example, all catalog items in *us_ncs_Spaces.xml* would in almost all instances have the <*Catalog Item Type="Space"*. In addition, the name assigned to individual items must be unique for the given Catalog Type.

Although it is not generally recommended, it is quite possible that someone might more than one Catalog Type in a single XML file. However, if that was done, then it would be recommended that they are strongly related. An example might be that <*Catalog Item*

Type="Red Spaces", <*Catalog Item Type="Blue Spaces"*, and <*Catalog Item Type="Yellow Spaces"* might all be created in a one file named *us_ncs_ColorSpaces.xml*.

The individual Property values for each Catalog item are nested inside the <*Defaults*> tag. The <*Property* tag contains tokens for *definition*, *name*, *value*, and sometimes, as in the following tag, *parametric name*.

```
<Property definition="us_ncs_HandRails"
name="us_ncs_HandRails/Dimensions/@HandRailLength" parametricName="HandRailLength"
value="9:0"/>
```

Conclusion

There are certainly other ways to organize the dataset files other than ones presented here. However they are organized one key principle should be to keep the local site customizations separate from the Bentley delivered datasets. The second important concept would be to enforce consistency in philosophy.

Certainly one concept not discussed here is to use the configuration files, such as *Triforma.ucf* or ArchProjectTemplateNcsUS.pcf to define additional search paths for the DG_DEF. Using this method addition directory folders can be used to segregate the data in was analogous to those presented here. Bentley suggests this method in at least one place in their documentation.

The only problems that will be encountered in the separate folder concept will be the file I have titled the Special Purpose dataset files: *us_ncs_enumlookups.xml*, *us_ncs_displaynames.xml*, *us_ncs_catalogtypeexts.xml*, and *MRULookups.xml*. Those will require manual editing to segregate Bentley as delivered dataset information from site customizations. However, this is true in either situation.

If there is a problem with this alternate method it is that additional modifications to *config* files are required in order to define the additional DG_DEF paths.