Customized Editing and Data Display in ArcInfo 8

By Todd Baltes, Software Developer

Editor's note: *This article addresses the needs of developers who desire an efficient editing environment. The author highlights the benefits of building custom editing tools and describes custom object inspectors, custom tools and commands for managing geodatabase data, and the management of nonfeature class geodatabase data.*

With the introduction of ArcInfo 8, the GIS user has unlimited possibilities for data model and GUI customization. ArcInfo 8 is quickly proving itself to be a very powerful, full-featured GIS. The development of the geodatabase as the storage mechanism for ArcInfo data introduces new challenges and opportunities to the technical community. Along with the standard data editing and display tools provided within ArcInfo 8, there are opportunities to create custom tools and methods for editing, displaying, and managing data within a geodatabase.

Why Customize Data Editing and Display?

A quick look at the ArcInfo 8 system reveals some of the standard tools provided for management and display of geodatabase data. In ArcMap, an object inspector, or properties editor, displays and allows editing of the attributes of selected features. Labels can be established to display desired geodatabase information for features in ArcMap as well. In ArcCatalog, the preview tab can be used to summarize the contents of a geodatabase. In addition to these standard tools, ArcInfo 8 provides the user with many ways to customize data editing and display.

With the standard editing tools that are available, what need is there for customization of this environment? Custom editing tools can be used to accomplish tasks such as displaying a subset of the attributes of a feature or displaying a different representation of an attribute value such as an image. Also, users with an existing information system (IS) implementation may need to incorporate the display or editing of data that resides outside of the ArcInfo geodatabase. Customization might include

- Limiting potential editing errors by omitting certain attributes from the display
- Enhancing the editing experience by providing additional tools for the selection or display of values
- Adding a customized GUI to fill related attributes within a feature
- Performing additional validations of data when editing
- Displaying external data from other IS systems within an organization or other representations of attribute data

In addition to the functional improvements that customized editing and data display can provide, there are other benefits to be gained from customization. It can provide a cost-effective solution to data integration issues by allowing data that exists outside of the ArcInfo 8 environment to be used from within ArcInfo 8 through customized editing tools. A customized editing environment can increase user-effectiveness by providing project-specific functionality. The ArcInfo 8 environment provides many methods for customizing the editing, display, and management of data—custom object inspectors, class extensions, and editor extensions and customization of data editing and displaying.

The Object Inspector (AKA Properties Editor)

The object inspector (or properties editor) is one of the primary geodatabase editing and display tools in ArcMap. The standard object inspector provides basic editing capabilities and displays all the attributes of selected features as well as related objects. Values can be changed simply by clicking on the display and entering a new value. This behavior, while generally beneficial, may be inappropriate if users should not edit, perhaps not even see, all of the attributes associated with a feature. The ArcInfo 8 platform provides a mechanism for implementing custom display and editing capabilities through the use of a customized object inspector.

To create a custom properties editor for a particular application, create a custom implementation of an object inspector. An object inspector is simply a COM object that implements two ArcInfo-defined interfaces: IObjectInspector and IClassExtension. These two interfaces provide the mechanism for the ArcInfo application software to interact with the custom object inspector.

The custom implementation of the IObjectInspector interface specifies the behavior of the GUI component(s) that represent the custom properties editor. This interface contains the following methods that must be implemented: Clear, Copy, hWnd, and Inspect. The Clear method is used to clear the custom object inspector in preparation for inspection of the next feature. The implementation of the Copy method determines how the fields are copied into the active feature from a passed-in source row. The hWnd method is used to return a handle of the custom object inspector to the ArcInfo application software. The Inspect method is used to perform the inspection, or display, of the selected feature(s). The UML representation of the IObjectInspector interface is shown in **Figure 1.**

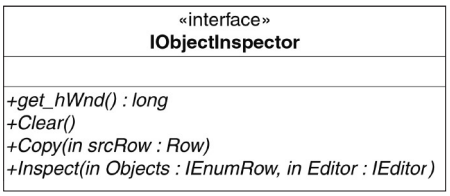

Figure 1

The hWnd method is the hook into the customized object inspector implementation. The customized object inspector implementation should return the hWnd of a GUI component that will be used for the object inspector. This may be the hWnd of a form, grid, picture box, or other GUI component. Passing the hWnd of this GUI component allows the ArcInfo 8 application to use the customized GUI component instead of the object inspector implementation provided by ArcInfo. An example of the Visual Basic code used to pass the hWnd of a GUI component to the ArcInfo 8 application software is shown in **Figure 2.**

```
  Private Property Get
IObjectInspector_hWnd() As
esriCore.OLE_HANDLE

    'Set the hWnd of a Grid to
display as the object inspector

    IObjectInspector_hWnd =
pGrid.hWnd
  End Property
```

Figure 2

The Inspect method implementation defines the display behavior associated with the custom object inspector. Use this method to define the ways that attribute information is displayed. Some customized behaviors that could be placed in this method include displaying images rather than attributes, looking up external data and displaying it instead of the ArcInfo attributes, or limiting the number of attributes that will be displayed by the object inspector.

Note that additional GUI components can be used in association with the custom object

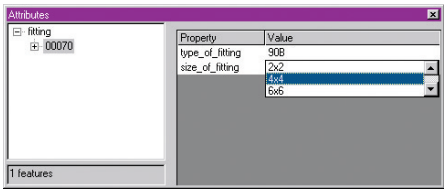

Figure 3

inspector. Events supported by the GUI components can be used within the customized object inspector implementation. For example, intercepting the Click event of a grid object could cause a list box to display domain values for the attribute being edited. Other GUI components can be used to enhance the editing process. Using a customized object inspector, elaborate input schemes can be implemented. An example of a custom object inspector (properties editor) that displays a subset of the attributes and uses a list box to display values from a domain is shown in **Figure 3.**

Class Extensions

The object inspector also implements the IClassExtension interface. This interface allows the ArcInfo application to identify and use the customized object inspector. The IClassExtension interface contains only two methods: Init and Shutdown. The Init method is called to initialize the custom COM object, and the Shutdown method is used to clean up any references within the custom COM object when it is destroyed. While it is required for the customized object inspector to implement the IClassExtension interface, a customized implementation of the IClassExtension interface is a useful data management tool by itself. See **Figure 4** for a UML representation of the IClassExtension interface.

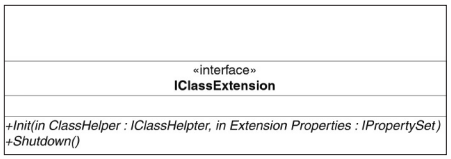

Figure 4

Class extensions are useful tools for managing both geodatabase data and external data within ArcInfo 8. As mentioned earlier, the IClassExtension interface provides the mechanism for the ArcInfo application to interact with a user-defined COM object that implements the IObjectInspector interface. Other interfaces are meaningful within the context of a class extension.

The IRelatedObjectClassEvents interface can be used to manage related feature data or external data. This interface contains methods that are called by the ArcInfo application software in response to events that occur in the editing environment. The IRelatedObjectCreated method must be implemented by the IRelatedObjectClassEvents interface. Implementation of this method allows the user to be notified whenever a related object is created. Information provided by this event can then be used to add, delete, or modify feature class data, object class data, or external data. The UML representation of the IRealtedObjectClassEvents interface is shown in **Figure 5.**

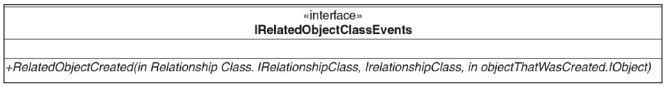

Figure 5

In addition to the IRelatedObjectCreatedEvents interface, custom event handling interfaces can be implemented, typically within the context of a custom feature. When a custom feature is implemented, it can query the class extension implementation to see if a particular interface is supported and call the methods in that interface if needed.

Two important aspects of class extension implementations need to be emphasized. First, class extensions are COM objects, and as such, can implement multiple interfaces. This allows a class extension to implement both display (and editing) behavior, as well as data management behavior during specific events within the editing environment. In addition, it is important to emphasize that class extensions are implemented at the class level, and every class or feature can have a different class extension. Although the customized behavior can be implemented strictly at the class feature level, it is also possible to implement some of the customized behavior as functionality that is shared between the class extension implementations.

Using Class Extensions

Class extension implementations allow the ArcInfo 8 application to execute the editing, display, and data management customizations. To use class extensions effectively, it is important to understand how the ArcInfo 8 application "knows" how to use these customized implementations. Class extensions are stored as part of the geodatabase. The CLSID value for the custom COM object is registered in the geodatabase, allowing the object access to the custom COM object via the CLSID. Object classes and feature classes within the geodatabase "look up" their associated class extension entry and create an instance of the COM object using the class extension's CLSID.

ArcInfo 8 provides an interface to manipulate the class extensions within the geodatabase. The IClassSchemaEdit interface supports methods that allow the class extension CLSID values to be altered within the geodatabase. Use the IClassSchemaEdit interface and its AlterClassExtensionCLSID method to change the class extension CLSID value for a feature or object. This is the proper method for setting a class extension CLSID within the geodatabase. When writing custom applications that alter the class extension CLSID values, a reference to the IClassSchemaEdit interface can be obtained from an IObjectClass instance.

One approach to managing class extensions is to use custom commands within the ArcMap environment. Implementing custom commands that use the IClassSchemaEdit interface AlterClassExtensionCLSID method could allow the user to select different property editors for the features within a model and create one set of object inspectors designed specifically for display and another set designed specifically for data editing. Many different editing customizations could be implemented for a single feature. When the class extension is changed within the geodatabase, the ArcInfo application software will create an instance of the new COM object to use as the class extension.

Another option available is to write a custom application to manage class extensions

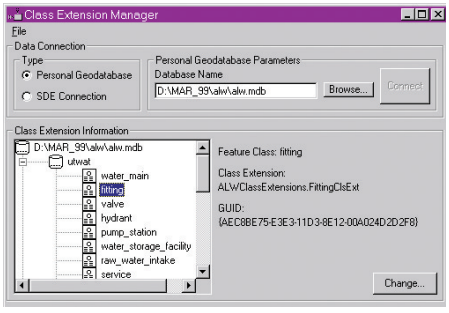

Figure 6

independently of the editing environment. An application can be created to run outside of the ArcMap environment. This application can properly set the class extension using the AlterClassExtensionCLSID method for any object class (or feature class) within a geodatabase. This type of tool can be quite useful in an environment where many customizations will be

Continued on page 30

Customized Editing and Data Display in ArcInfo 8

Continued from page 29

needed because it provides a quick interface for testing new implementations or temporarily disconnecting the class extension from the ArcMap application. An example of what this application might look like is shown in **Figure 6.**

Editor Extensions

One other option for managing geodatabase data or external data is with a customized editor extension implementation. An editor extension allows customization to be placed at specific events within the editing environment. While this type of customization is similar to the class extension implementation, it differs in one major respect—the editor extension responds to events that occur at the editor level, not the object or feature level. This allows for customizations that can be shared by features within the model or that are global to all features within the model.

A customized editor extension is a COM object that implements the IEditor interface. This interface is quite simple—it contains the Name, Startup, and Shutdown methods. The Name method simply returns the name of the custom editor extension. The Startup method defines what happens when the extension starts. The Shutdown method defines what happens when the extension shuts down or is destroyed.

To customize the editing behavior in response to events from the editor, you must save the initialization data that is passed to the Startup method. The parameter is a reference to the editor. A member variable must be defined to store this reference (as IEditor). It is also necessary to create a reference to the IEditEvents interface that the editor supports.

The IEditEvents interface intercepts events and allows the custom code to execute. In Visual Basic, the definition for the reference to the IEditEvents interface is achieved using the "WithEvents" clause in a variable definition of type Editor. Set the IEditEvents reference equal to the IEditor reference during the Startup method so the editor extension can listen for events generated by the editor object.

The editor extension can intercept many different editor events such as OnStartEditing, OnCreateFeature, OnChangeFeature, OnDeleteFeature, and AfterDrawSketch. These events occur at the editor level. Check the type of feature(s) involved when responding to an editor event if the customization pertains to a particular feature type (or some subset of all the feature types).

These events are a good place to put custom code for managing external data. For example, this code might create a record in an existing database outside the ArcInfo environment whenever a particular feature type was created or deleted during editing. In this case, it may be beneficial to simply track the editing changes during the OnCreateFeature and OnDeleteFeature methods, saving the actual creation and deletion of the external data for the OnStopEditing event. The OnStopEditing event passes a parameter to indicate whether or not the edits from the session should be saved. An example of the Visual Basic Code needed to use Editor events in an Editor extension is shown in Figure 7.

```
  Implements IExtension

  Private m_pEditor As esriCore.IEditor

  Private WithEvents m_pEditorEvents As esriCore.Editor

  Private Sub IExtension_Startup(initializationData As
Variant)

    ' Save a reference to the editor

    Set m_pEditor = initializationData

    ' Hook into the editor's event model

    Set m_pEditorEvents = m_pEditor

  End Sub

  Private Sub m_pEditorEvents_OnCreateFeature(ByVal obj As
esriCore.IObject)

    'Now, do something whenever a feature is created

    MsgBox "Feature Created!"

  End Sub
```

Figure 7

Conclusion

While ArcInfo 8 offers the user many standard tools for data display and editing, there are also many options available to create a customized editing environment. While these techniques are only a few of the possibilities available when creating a customized ArcInfo environment, there is much that can be accomplished using them including the ability to

- Create custom object inspectors for specific applications or users.
- Manage external data from within the ArcInfo 8 environment.
- Provide an application-specific look and feel.

Many other types of customizations are also possible using the techniques covered here. Customized editing and display of data unlocks many benefits and capabilities of ArcInfo 8. These customizations can provide cost-effective solutions to issues such as data integration and user training.

Acknowledgements

The author thanks Brian Goldin at ESRI for his insight and help; Anthony Thorpe, Steve Straka, Mike Watry, Brad Barnell, and all the others at Analytical Surveys, Inc., who have reviewed the prototypes and implementations that served as the background for this article as well as reviewing this article and contributing their ideas; and Kim Baltes for her help in editing this article.

For more information, contact

Todd Baltes
Software Developer
Saligent Software, Inc.
E-mail: todd@baltes.net

About the Author

Todd Baltes attended University of Wisconsin–Waukesha and has been involved with GIS since 1988. His primary focus has been application development, customized application extensions, and application design consulting.