autodesk®

RELEASE 5

Autodesk

MapGuide®

DEVELOPER'S GUIDE

15505-010000-5060 July 2000

1 2 3 4 5 6 7 8 9

Contents

Chapter 1 Introduction 5

What Can Custom Autodesk MapGuide Applications Do? 6
- View Maps 6
- Query and Update Data 6

What Do I Need to Know Before I Begin? 7
- Autodesk MapGuide 7
- Programming and Scripting Languages 7
- Your Audience 8

Chapter 2 Developing with the Viewer API 9

The Autodesk MapGuide Viewer API 10

Creating an Autodesk MapGuide Viewer Application 11
- Displaying the Map 11
- Accessing the Map Programmatically 15
- Working with Autodesk MapGuide Viewer Events 18
- Detecting and Installing the Viewer onto Client Systems 21
- Handling Display Refresh and the Busy State 23
- Handling Errors 26
- Accessing Secure Data 26

Chapter 3 Viewer API Examples 29

Performing Common Tasks with the API 30
- Counting Layers 30
- Listing Layers 31
- Adding a Layer 32
- Linking Layers 33
- Retrieving Keys of Selected Features 35
- Retrieving Coordinates of a Selected Feature 37
- Invoking Select Radius Mode 39
- Toggling a Layer On and Off 40
- Zooming In on Selected Features 41
- Counting Map Features 42

Customizing the Printout 44
Advanced Applications 49
Custom Redlining Application 49
Municipal Application 53
Facility Management Application 69

Chapter 4 Using Reports to Query and Update Data Sources 77

Autodesk MapGuide Reports 78
How Reports are Generated 78
Specifying the Report Script 78
The Request 79
Launching the Report 79
Introducing ColdFusion and ASP 79
Creating Report Scripts with ColdFusion 80
Example—Listing File Contents with ColdFusion 81
Example—Querying and Displaying Data via the Map 83
Example—Modifying a Database via the Map 94
Creating Report Scripts with ASP 102
Summary of ASP Objects, Components, and Events 103
Example—Listing File Contents with ASP 104
Example—Querying and Displaying Data via the Map 108
Example—Modifying a Database via the Map 119

Chapter 5 Using the SDF Component Toolkit to Modify Spatial Data 129

About the SDF Component Toolkit 130
Toolkit Objects 130
Status Codes 132
Enumerated Constants 133
Working with SDF Files 133
Indexing 133
Editing 134
SDF Pitfalls 134
Performing Common Tasks with the Toolkit 135
Updating SDF Files via the Map 136
Visual Basic Examples 146
Converting To an SDF File 146
Getting Information about an SDF File 153
Copying an SDF File 161

Index 167

Introduction

In This Chapter 1

- What can custom Autodesk MapGuide applications do?
- What do I need to know before I begin?

The Autodesk MapGuide® product suite provides you with all the tools you need to create, publish, and display maps and associated attribute data over the web. The *Autodesk MapGuide Developer's Guide* is a complete guide to Autodesk MapGuide's customization and development features. This chapter defines Autodesk MapGuide application development, describes how to use this manual, and lists additional sources of information.

What Can Custom Autodesk MapGuide Applications Do?

An Autodesk MapGuide application can be as simple as an HTML page that displays an embedded map window file (MWF), or it can be as complex as a CGI application, coded in C++, that modifies data files on the server and refreshes the browsers of everyone viewing the map. Usually it is something between the two, such as a map embedded in a web page with buttons and other controls on it that interact with the map.

Following is more detail about the types of tasks your Autodesk MapGuide application can do.

View Maps

The most common development goal is to allow Autodesk MapGuide Viewer users to view and interact with maps. You can do this by embedding a map in an HTML page, in which case the Viewer runs within the user's web browser to display the map, or you can run the Viewer from within a stand-alone application that you create. With either approach, you will use the Viewer API to interact with the map. For example, you might create a button that refreshes a map or add text boxes that allow the user to add data to the map.

Query and Update Data

Beyond viewing maps, users want to retrieve data to answer questions. This includes selecting map features and running reports on them, such as selecting power poles and seeing when they were last serviced. You set up these reports using Allaire ColdFusion®, Microsoft Active Server Pages™ (ASP), or another server-side scripting language. Additionally, you can use these scripts to enable the user to update the data. For example, you could display the date of last service in a text field, where the technician in the field could update it. Your script would then take the technician's date and update the source database, so that all other technicians viewing that power pole on the map would see the new date. This is discussed in Chapter 4, "Using Reports to Query and Update Data Sources."

You can also enable users to mark up the maps to edit the spatial data, such as correcting the location of a fire hydrant by drawing its correct location on the map. This process is called redlining. Autodesk MapGuide Release 5 provides APIs that allow you to add redlining functionality to your map. You

can then create a server-side script that retrieves the redlining data, processes it, and updates the source data. This process is described in Chapter 2, "Developing with the Viewer API." If your source data is in SDF files, you can use the Autodesk MapGuide SDF Component Toolkit to update the SDFs directly when redlining. This process is described in Chapter 5, "Using the SDF Component Toolkit to Modify Spatial Data."

What Do I Need to Know Before I Begin?

Before you can begin developing with Autodesk MapGuide, there are several requirements you must meet.

Autodesk MapGuide

You need to be very familiar with Autodesk MapGuide. In particular, you should read the first few chapters of the *Autodesk MapGuide User's Guide* to make sure you understand the product, especially emphasizing the following sections:

- Chapter 2, "Understanding Autodesk MapGuide." Read this chapter carefully, with particular attention to the sections on how the components work together, application development components, and what application development is.
- Chapter 3, "Designing Your System." Pay particular attention to the sections on security, architecture and performance, and choosing a viewer/browser environment.

The more you understand about the Autodesk MapGuide components and how they work together, the easier it will be for you to comprehend the examples in this book and come up with unique solutions on your own.

Programming and Scripting Languages

To build custom Autodesk MapGuide Viewer applications, server-side scripts, and reports, you will need to be able to perform the following tasks:

- Create HTML pages with embedded Java™, JavaScript, JScript, or VBScript code that accesses the objects of the Autodesk MapGuide Viewer API. For example, you might create an HTML form containing a button that turns a layer on and off, or a drop-down list that selects map features.
- Create custom reports with ColdFusion, Active Server Pages (ASP), or another third-party application.

- Create server-side applications with Autodesk MapGuide SDF Component Toolkit (to dynamically update SDF files posted on an Autodesk MapGuide Server) or with another tool (to process redlining data and update your data sources).

Your Audience

As with all development, the most important aspect of designing your application is asking yourself, "What do my users need?" Talk to the people who will be using your application and find out how they will be using it. What tasks will they want to perform? Will they need redlining? Are they computer savvy, or will you need to guide them through basic usage of your application? Do they have much domain knowledge (for example, are they all electricians, or will some of them be administrative assistants)? It's critical that you find out what tasks your users will need to perform, as well as their knowledge of those tasks.

If you want to provide information about your application that users can readily access, you can develop your own set of help pages. You can then set up the map to point to your help instead of the default Autodesk MapGuide Viewer help when users click the Help button or access help from the popup menu. For more information, refer to the *Autodesk MapGuide Help*.

Developing with the Viewer API

In This Chapter

2

- The Autodesk MapGuide Viewer API
- Creating an Autodesk MapGuide Viewer application

The Autodesk MapGuide Viewer API allows you to customize the way in which a Viewer user interacts with your map. For example, you can add buttons and other controls that interact with the Autodesk MapGuide Viewer and the map. You can also create a stand-alone version of the Autodesk MapGuide Viewer that displays maps without the use of a web browser. This chapter introduces the API and describes the main tasks involved with creating a client-side application.

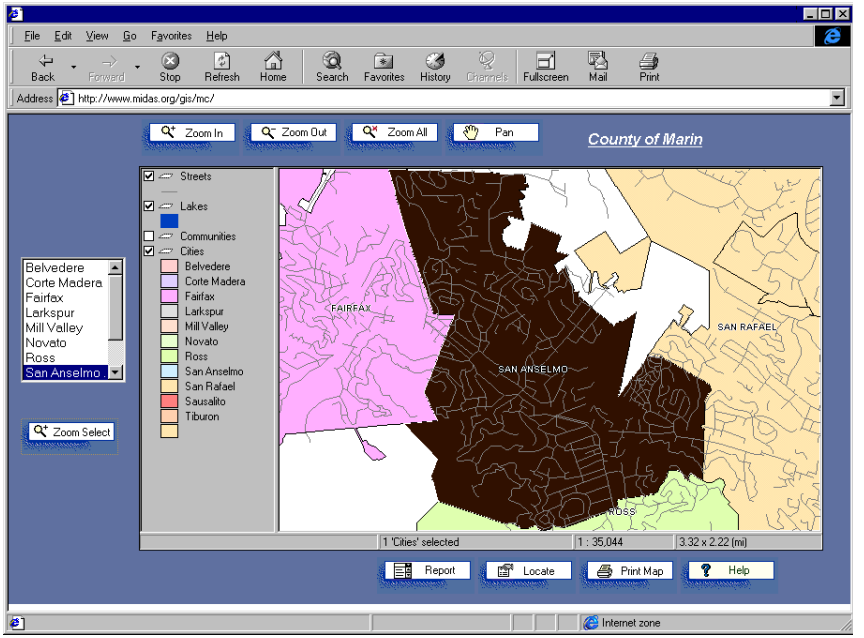
The Autodesk MapGuide Viewer API

The Autodesk MapGuide Viewer comes in three different versions: a Microsoft ActiveX Control, a Netscape Plug-In, and a Java edition.

Each Viewer exposes an API that contains all of the Viewer functionality. An *API* (application programming interface) is a set of classes, methods, and events within an application that other applications can access. In this case, the Autodesk MapGuide Viewer API enables you to create your own web applications that interact with the Autodesk MapGuide Viewer.

For more info...
Take a look at the sample custom Viewer applications at *www.autodesk.com/mapguidedemo* to get a better idea of the types of applications you can create.

For example, you could create an application that displays a map in one frame and a form in another. In the form, you might have controls such as buttons and list boxes that use API methods to alter or redraw the map. Or you might put the map and controls on a single page, as shown in the following illustration. This application consists of a map, a form listing map features, and a number of custom image buttons, all lined up in an HTML table. Users can select a city from the list box, and then click a button to zoom to that city.



Sample application with custom buttons

You could also code your application to update the form, display status information, or change the appearance of buttons as users select or double-click specific features on the map. This two-way interaction between the map and controls on the web (HTML) page allows you to create very powerful applications.

Keep in mind that an application in this context is a web page containing one or more maps, each of which is displayed in a separate instance of the Autodesk MapGuide Viewer, and any additional items on the web page, such as frames, controls, graphics, and so on. Therefore, all of your application code is contained in the web page itself—you use one of the supported languages (see "Choosing a Viewer/Browser Environment" in the *Autodesk MapGuide User's Guide*) to write the code for your application inside of the HTML code in your web page. The rest of this chapter describes this process in detail. For complete information about using the Autodesk MapGuide Viewer API, refer to *Autodesk MapGuide Viewer API Help*, available from the Autodesk MapGuide CDs and the Autodesk MapGuide documentation page (*www.autodesk.com/mapguidedocs*). The Viewer API Help provides descriptions of all of the Viewer API objects, methods, properties, and events, and it includes sample applications that you can use to get a quick start.

Creating an Autodesk MapGuide Viewer Application

This section describes the essential tasks you need to perform when creating a Viewer application, as well as some key concepts you need to keep in mind.

Displaying the Map

As described in the previous section, you create all of your application code in the same web page where your map is displayed. Therefore, the first step is to learn how to add the map to the web page. If you are creating a simple application, such as a web page with nothing but a map and text, this step might be all you need to know.

An important concept to keep in mind is that when you add a map to a web page, you are not inserting an empty Autodesk MapGuide Viewer in which you can open maps. Rather, you are adding a specific map to the web page, and the user's copy of the Autodesk MapGuide Viewer runs automatically to display it. This is an important distinction, because it means that your users can view the same map with one of the four Autodesk MapGuide Viewers, depending on which ones you support. The Viewers you support depend on

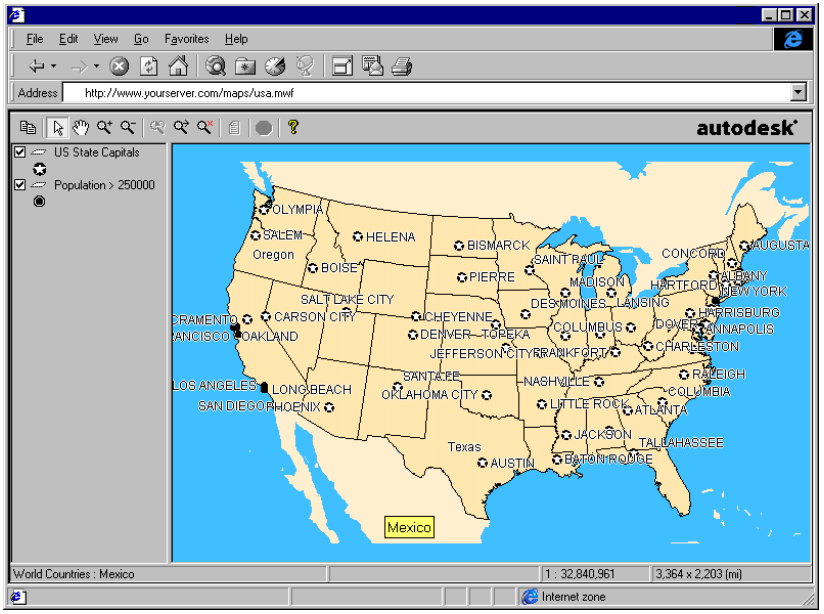
which scripting language you are using and which browsers your users have (see "Choosing a Viewer/Browser Environment" in the *Autodesk MapGuide User's Guide),* as well as how you add the map to the web page. The following techniques explain how to add the map to support the different Viewers.

For the ActiveX Control and the Plug-In

To display your map with the ActiveX Control and/or the Plug-In, you can either create a link to the map or embed the map as follows:

Linking to a Map

When you link a map, you create a link to the map from an HTML page, so that when the user clicks that link, the map displays full screen. Note that the browser displays the map by itself, not as part of an HTML page.



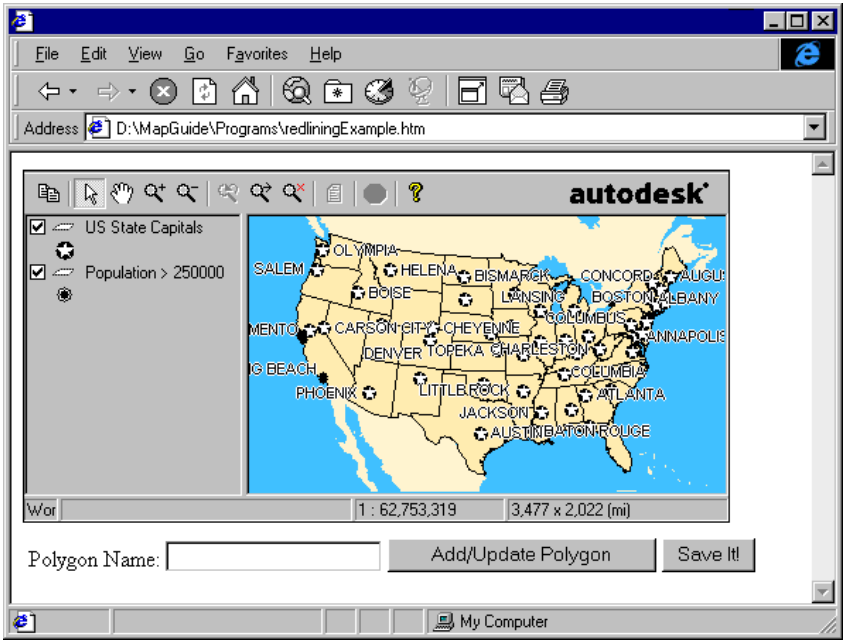
Linking to a map

You create the link just like any other link in HTML, using the <A> tag with the HREF parameter. Set the HREF value to the URL of your Autodesk MapGuide Server, along with the maps directory alias and the MWF (map) file. For example:

```
<A HREF="http://www.yourserver.com/maps/usa.mwf">United States Map</A>
```

Embedding a Map

When you embed the map in an HTML page, the map displays with the rest of the information on that page when the user visits the page.



Embedding a map

To embed the map, use the EMBED (for Netscape Navigator) or OBJECT (for Microsoft Internet Explorer) tag in the page. For more information and examples, refer to "Embedding Your Map" in the *Autodesk MapGuide Viewer API Help*.

To ensure that both Netscape Navigator and Internet Explorer can access the map, you use both tags. For example:

```
<OBJECT ID="myMap" WIDTH=300 HEIGHT=200
   CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.mapguide.com/maps/usa.mwf">
   <EMBED SRC="http://www.mapguide.com/maps/usa.mwf" NAME="myMap"
      WIDTH=300 HEIGHT=200>
</OBJECT>
```

Refer to "Autodesk MapGuide Viewer URL Parameters" in *Autodesk MapGuide Viewer API Help* for a list of the parameters that control the way the map is displayed when it is linked to or embedded in an HTML page. Be sure that the values you use are the same for both the OBJECT and EMBED parameters.

For the Java Edition

To display a map in the Java edition, you need to use the <APPLET> tag in your HTML page. You cannot create a link to a map using the Java edition; the map must be embedded in the page.

To display your map with the Java edition

1. Start with the standard HTML <APPLET> tag.
2. Set the CODE parameter to MGMapApplet.
3. Set the VALUE parameter to the URL of your Autodesk MapGuide Server along with the maps directory alias and MWF file. For example:

```
<HTML>
<HEAD>
<TITLE>Autodesk MapGuide Java Edition Example</TITLE>
</HEAD>
<BODY>
<CENTER>
<H2>Simple Invocation of Installed Autodesk MapGuide Applet</H2>
<APPLET WIDTH=300 HEIGHT=200 ALIGN="baseline"
   CODE="com/autodesk/mgjava/MGMapApplet.class">
   <PARAM NAME="myMap" VALUE=
        "http://www.mapguide.com/maps/usa.mwf">
</APPLET>
</CENTER>
</BODY>
</HTML>
```

For the LiteView Extension

Now in Release 5 of Autodesk MapGuide, users can view maps as static raster images using the new Autodesk MapGuide LiteView Extension. This is useful when users need only to display the map and do not need the more advanced queries and other functionality of the other three Viewers. Best of all, users do not have to download a Viewer to view your maps; the LiteView Extension displays the map to any visitor to your web page.

To display a map using the LiteView Extension, you send a URL request that returns the map displayed as a raster image file in the browser. Note that you do not use the Viewer API with the LiteView Extension. For complete information on implementing the LiteView Extension, refer to the *Autodesk MapGuide LiteView Extension Developer's Guide*.

Accessing the Map Programmatically

For more info...
Read "Getting Started" in the *Autodesk MapGuide Viewer API Help*.

Once your map is embedded in a web page, you can start to write your custom application. The first step in that process is learning how to access the map from your code. You do this by creating an instance of the MGMap object; the code you write uses that instance when it calls methods in the Viewer API.

Although Netscape and Internet Explorer expose the map object at different levels in their object hierarchies, you can write a simple function that checks the user's browser type and returns an instance of the map object using the method required by that browser. Once you've obtained the map object, the code for either browser will in most cases be identical.

Note Because Internet Explorer automatically recompiles JavaScript code into its native JScript, you can write JavaScript code that works with either browser.

Suppose you embedded your map and named it "myMap". In Netscape Navigator, the map object called "myMap" is exposed by the document object and can be accessed from JavaScript in one of the following ways:

```
document.myMap              // one way...
document.embeds["myMap"]    // another way...
```

In Microsoft Internet Explorer, the map object called "myMap" is exposed by the window object and can be accessed from JavaScript in one of the following ways:

```
window.myMap                // one way...
myMap                       // another way...
```

The easiest way get around these differences is to write a function that checks the browser type and returns the appropriate map object; that function can then be called as needed by the rest of the code in your application.

If your embedded map has the name "myMap", the code to access the map on a simple, frameless HTML page would look like this:

```
<SCRIPT LANGUAGE="JavaScript">
function getThisMap()
{
   if (navigator.appName() == "Netscape")
      return document.myMap;
   else
      return window.myMap;
}
</SCRIPT>
```

If your application has multiple HTML frames, the code to access the same map in a frame called "Left" would look like this:

```
<SCRIPT LANGUAGE="JavaScript">
function getThisMap()
{
   if (navigator.appName == "Netscape")
      return parent.Left.document.myMap;
   else
      return parent.Left.myMap;
}
</SCRIPT>
```

Note We chose the name "getThisMap()" for our function, but the name can be anything you want, as long as it follows JavaScript naming conventions. As you look at the source of other Autodesk MapGuide applications, you might notice that the name "getMap()" is often used; this name should not be confused with MGMapLayer.getMap(), a predefined Viewer API method.

After this function is defined, any other JavaScript method can simply call getThisMap() to retrieve the map object. For example, you could create a variable to represent the map, and then use getThisMap() to set the value of that variable:

```
var map = getThisMap();
```

You could then apply methods to that variable to work with the map. For example, the following function displays an Autodesk MapGuide report called "Parcel Data":

```
function runReport()
{
   var jb_map = getThisMap();          // assign map to variable
   jb_map.viewReport('Parcel Data');   // call method from variable
}
```

Or you could bypass the variable assignment and use getThisMap() directly:

```
function runReport()
{
   getThisMap().viewReport('Parcels'); // use getThisMap() return value
}
```

Because getThisMap() has an MGMap object as its return value, you can use the function in place of MGMap, accessing that object's methods or properties as needed.

About the Java Edition

For more info...
Read "Implementing the Java Edition" In the *Autodesk MapGuide Viewer API Help*.

The process described above works for the Plug-In, the ActiveX Control, and the Java edition—the object hierarchy is determined by the browser, not by the version of the Autodesk MapGuide Viewer. Note, however, that JavaScript and JScript do not behave uniformly across all browser/operating system combinations. In particular, Internet Explorer has the following limitations:

- Internet Explorer 4.0 for Mac OS does not support JavaScript. It supports JScript, but JScript cannot control a Java applet.
- Because Internet Explorer exposes applets as COM objects instead of Java objects, API methods that pass observer objects will not work. For example, digitizePoint requires an instance of the MGDigitizePointObserver object. Therefore, Internet Explorer wouldn't be able to access digitizePoint or any other methods that pass observer objects as arguments, including the following MGMap methods: addMapLayer and addMapLayers, all of the digitize methods, and viewDistance and viewDistanceEx. If you need to use any of these methods, you will need to implement the Java edition of the Autodesk MapGuide Viewer from Java instead of JScript.

The following table clarifies the various levels of support that JavaScript/JScript provide across different configurations.

JavaScript/JScript support with each operating system/Viewer/browser combination

Operating System	Version of the API	Level of Support
Windows	Plug-In on Netscape	Full support
Windows	ActiveX Control on Internet Explorer	Full support
Windows	Java edition on Internet Explorer	Only methods that do not pass observer objects as arguments
Windows	Java edition on Netscape	Full support
Solaris	Java edition on Netscape	Full support
Mac OS	Java edition on Internet Explorer	Cannot control Java applet

Therefore, if you need to support the largest number of browsers and operating systems, you should simply embed the Java edition in the page and not use JavaScript or JScript to extend the functionality of the page to interact with the map, or you will need to write your application in Java. This is particularly important if you need to support Macintosh users, as they will need to use the Java edition with Internet Explorer, which doesn't have full JScript support.

For more information about the Java edition, refer to "Working with Java" in the *Autodesk MapGuide Viewer API Help*.

Now that you have displayed the map and created an instance of it, you are ready to begin coding. However, there are some key concepts you need to keep in mind. The rest of this section describes these concepts.

Working with Autodesk MapGuide Viewer Events

Just as the web browser has events that are triggered in response to actions within the browser, Autodesk MapGuide has its own events that are triggered by actions within the Viewer. For example, if the user selects a feature on the map, the onSelectionChanged event is triggered. There are also events for when the mouse double-clicks, the map view changes, and more.

Event Handlers and Observers

Events are useful because you can write code that is executed only when certain events occur. For example, if the user clicks a point on the map, you might want to call a function that updates text boxes with the coordinates for that point (see "Updating SDF Files via the Map" on page 136 for an example of this). This type of function, which works only in response to an event, is called an *event handler*.

Before you can capture events and call the event handler, however, you need to ensure that you have *event observers* set up. Event observers act as the link between the event and your event handler; they are triggered when an event occurs and then call the event handler in response.

Event Observers in Internet Explorer vs. Netscape Navigator

This all sounds fairly straightforward: an event occurs, it's picked up by an observer, which then calls the event handler function that does something in response to the event. The complication is that Netscape Navigator and Internet Explorer use event observers differently, so if you want to support both browsers, you need to write code for both. For Netscape, Autodesk MapGuide provides an observer applet that you embed in your application using the APPLET tag. For Internet Explorer, you create your own observer by writing a few lines of VBScript code that tell Internet Explorer the name of the event handler. This is demonstrated in the example on page 20.

Naming Conventions

There is one more place where things get complicated: when you call an Autodesk MapGuide method that triggers an event (such as digitizePoint), you pass the Autodesk MapGuide observer as a parameter if the browser is Navigator, but you do not pass the VBScript observer method if the browser is Internet Explorer. This is because Internet Explorer counts on a standard naming convention for event observers: when an event occurs, Internet Explorer looks for a VBScript method with the name of the object in which the event occurred (in this case, the map) followed by the event (such as mapname_onDigitizedPoint in response to an onDigitizedPoint event). If it finds such a method, it treats it as the event observer, reads it to find out what function to call next (the event handler), and then calls that event handler. Thus, Internet Explorer knows how to find the observer without being passed its name. Netscape does not have this same logic—you have to pass the name of the observer to get it going.

At this point, you might be wondering how the Autodesk MapGuide observer can possibly know where to send the event, that is, how does it know the name of your event-handler function? This is where the standard naming convention comes in again. The Autodesk MapGuide observer assumes that your event handler function is going to use exactly the same name as the event. Therefore, to make the observer work properly, always name your event handler function the same name as the event itself. Also note that both the observer applet and the VBScript observer you wrote can point to the same event handler function—at this stage, you do not need to do anything extra to make it work for both browsers.

An Example

Let's say you want to support both Internet Explorer and Netscape. In your HTML page, you create the VBScript function that will act as the observer for the onDigitizedPoint event:

```
<SCRIPT LANGUAGE="VBScript">
//send onDigitizedPoint events from the ActiveX Control to the
//event-handling function
Sub map_onDigitizedPoint(Map, Point)
   onDigitizedPoint Map, Point
End Sub
</SCRIPT>
```

You also embed the observer applet for Netscape, *MapGuideObserver5.class*:

```
// ...if Netscape, embed event observer
if (navigator.appName == "Navigator")
{
    document.write("<APPLET CODE=\"MapGuideObserver5.class\"
    WIDTH=2 HEIGHT=2 NAME=\"obs\" MAYSCRIPT>");
    document.write("</APPLET>");
}
```

Notice that we've given the observer the name "obs". When you call this observer, you will refer to it as "document.obs", because in the Netscape object model, it is an object of the document. Be sure to copy the *MapGuideObserver5.class* file to the same directory as your HTML files, or this code won't work.

Now, let's say you have a button named "Digitize". This button is set up so that its onClick event (a browser event) calls a function you created called DigitizeIt(). The DigitizeIt() function calls the digitizePoint() method, a Viewer API method that waits for the user to click a point on the map and then captures that point. Because the digitizePoint() method requires an observer as a parameter if you're supporting Netscape, you write code for both:

```
function digitizeIt()
{
   if (navigator.appName == "Navigator")
      getMap().digitizePoint(document.obs);
   else
      getMap().digitizePoint();
}
```

So if the user is viewing the map in Netscape, the observer applet ("document.obs") is passed as a parameter. If the user is viewing the map in Internet Explorer, no observer is passed, because Internet Explorer will know that you have an event observer method waiting to observe this event. The browser waits for the user to click a point on the map, which triggers the onDigitizedPoint event. Then, one of the two observers picks up the event and tells the

browser what function to call next, namely an event handler function you named onDigitizedPoint. The onDigitizedPoint function then does whatever you want with the event, such as retrieving the coordinates of the point the user clicked.

Additional Information

There are two more tips that you need to know about events.

First, if you embedded *MapGuideObserver5.class* in a different frame or window from the function where you called it, you will need to specify the full window.frame.document.object name, such as "parent.mapframe.document.obs".

Second, there are some methods whose sole function is to allow you to set the observer for a specific event. For example, when the map view changes (such as in response to a user panning or zooming), how do you set the observer for the onViewChanged event? You create a function called onLoad (a browser event) and insert the setViewChangedObserver method to set the observer for the onViewChanged event. You can do this for each of the events you want to handle. Note that specifying an observer is not required for all events—just for the events you want to handle. Also note that although you will need to create a separate Internet Explorer observer for each type of event you want to handle, you can use the same instance of *MapGuideObserver5.class* for all events in Netscape Navigator.

To see a sample of how to handle events, refer to Examples in the *Autodesk MapGuide Viewer API Help*. Also in the Help, refer to "Setting up the Event Handler" in the Overview and "Events" in the Object Reference.

Detecting and Installing the Viewer onto Client Systems

For more info...
Read the "Getting Started" page In the *Autodesk MapGuide Viewer API Help*.

If users accessing your web site don't have an Autodesk MapGuide Viewer installed on their system, they need to download one in order to view the map you have displayed in the web page (unless you are using the LiteView Extension; see page 14). You can include code in your HTML file that automatically detects whether or not the user has the Viewer, and then either downloads it automatically or prompts the user to download it themselves.

First, add the map to the HTML page. Include the CODEBASE parameter to automatically detect whether the latest version of the ActiveX Control is installed (the CODEBASE parameter is ignored if the user is using Netscape Navigator). If the user has an older version, the latest version will be installed.

```
<OBJECT ID="myMap" WIDTH=300 HEIGHT=200
   CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D"
   CODEBASE="ftp://ftp.autodesk.com/pub/mapguide/ver5/
      viewer/en/mgaxctrl.cab#Version=5,0,0,0">
   <PARAM NAME="URL" VALUE="http://www.mapguide.com/maps/usa.mwf">
   <EMBED SRC="http://www.mapguide.com/maps/usa.mwf"
      NAME="myMap" WIDTH=300 HEIGHT=200
      PLUGINSPAGE="<http://www.autodesk.com/products/mapguide/
         vdwnload.htm>">
</OBJECT>
```

The ActiveX Control download is handled automatically for Internet Explorer users, but if a Netscape user doesn't have the Plug-In or doesn't have the latest version, you need to write additional code to prompt the user to download the Plug-In.

```
// Call this function on a page onLoad event or frameset onLoad event
function init()
{
   // For Netscape browsers, check for plug-in
   if (navigator.appName == "Netscape")
   {
      for(j=0;j<navigator.plugins.length;j++)
      {
         if (navigator.plugins[j].name == "Autodesk MapGuide")
            return;
      }
      // If the plug-in is not detected, display the message...
      displayDownloadMsg();
      return;
    }
   // If the Plug-In is installed, check the version by returning
   // the API version
   var version = getThisMap().getApiVersion();
    //If the API/Plug-In version is previous to 5.0,
   //display the message
   if (version < "5.0")
   {
      displayDownloadMsg();
      return;
   }
}
function displayDownloadMsg()
{
   // Display dialog box.
   msg = "You do not have the latest version of " +
         "Autodesk MapGuide Viewer. Do you want to " +
         "download it now? Click OK to download or Cancel" +
         "to proceed with your current Viewer."
   // If user clicks OK, load download page from Autodesk web site
   if (confirm(msg))
      top.window.location =
        "http://www.autodesk.com/products/mapguide/vdwnload.htm";
}
```

Handling Display Refresh and the Busy State

When the Autodesk MapGuide Viewer refreshes the display, it can cause errors in your application unless you take the correct steps to prevent them. You need to familiarize yourself with the way that the API is designed in order to understand how to code your application correctly.

Autodesk MapGuide Viewer enters a busy state whenever it refreshes the display. In this situation, the Viewer enters the busy state, which does not end until the data has been received from the server and the display is updated.

There are two important points to remember about the busy state:

- When your Autodesk MapGuide Viewer application calls an API method that causes a busy state, the Viewer can return control to the application and then go on to the next method while the Viewer is still in the busy state.
- Some methods in the Viewer API do not function correctly if the application calls them while the Autodesk MapGuide Viewer is in a busy state. Instead, the Viewer sets the error code in MGError to -1 for busy. The descriptions of the methods in the *Autodesk MapGuide Viewer API Help* clearly state which methods do not work during the busy state.

To avoid errors, you need to make sure that the Autodesk MapGuide Viewer is not in a busy state when your application calls one of these methods.

Your application is most likely to fail when it is about to call two or more API methods, the first one being an API method that automatically invokes a refresh, and the following one(s) being methods that don't work during the busy state. For example:

```
function selectAndZoomToPointObject(mgObj)
{
var mgMap = getThisMap();
var mgSel = mgMap.getSelection();
mgSel.clear();
mgSel.addObject(mgObj);
mgMap.zoomSelected();    // Busy state begins in zoomSelected()
mgMap.setWidth(5, "KM"); // Error occurs because setWidth() fails if
                         // called during the busy state
}
```

To avoid errors, you need to make sure that Autodesk MapGuide Viewer is not in a busy state when your application calls one of these methods. To do this, you can control display refresh using the autoRefresh flag, you can detect when a refresh is about to happen, and you can detect a change in the busy state. Each of these approaches is described in the following sections.

Controlling Display Refresh

You can ensure that the Viewer will not enter a busy state by controlling when display refreshes occur. The first step is to remember that display refreshes always occur in the following instances:

- The application calls an API method that requires an automatic refresh, and the API has the autoRefresh flag set to true. In the *Viewer API Help*, the methods requiring an automatic refresh are noted as such.
- Your application calls zoomGotoDlg, zoomGotoLocation, setUrl, refresh, or setAutoRefresh. These methods *always* invoke a display refresh, even if the autoRefresh flag is set to false.

To develop an application that executes smoothly, you need to prevent busy states from happening while the application calls methods in the API that don't work during the busy state. To do this, you need to disable the autoRefresh flag. Specifically, you need to set the autoRefresh flag to false immediately before calling the first method, then reset the autoRefresh flag to true and call the refresh method immediately after your application calls the other methods that don't work during the busy state. For example, take a look at this modified code:

```
function selectAndZoomToPointObject(mgObj)
{
   var mgMap = getThisMap();
   var mgSel = mgMap.getSelection();
   mgMap.setAutoRefresh(false); // Prevent busy state from happening
                                // when zoomSelected is called
   mgSel.clear();
   mgSel.addObject(mgObj);
   mgMap.zoomSelected();
   mgMap.setWidth(5, "KM");
   mgMap.setAutoRefresh(true);  // Reset the autoRefresh flag
   mgMap.refresh();             // Update the display
}
```

Note that simply setting the autoRefresh flag back to true does not refresh the map; you must also call the refresh method after resetting the flag.

AutoRefresh Flag Caveats

There are caveats you need to know when setting the autoRefresh flag to false. While autoRefresh is disabled, methods that would normally cause refreshes to occur do not, and the following types of operations may not work as expected:

- *Enumerating map features on dynamic layers after a pan or a zoom*. If your application tries to return the number of features on a dynamic layer prior

to a refresh, it will return the number that existed before the pan or zoom occurred.

- *Querying on or modifying selected features*. If your application performs queries or modifications on features on dynamic layers prior to a refresh, the features may not actually exist anymore, or additional features that were added to the selection may be missing.
- *Operations that require user interaction*. Methods such as digitizePoint and digitizeRectangle require users to click or drag the mouse for their input parameters. However, users will be positioning the cursor over a version of the map that is different from the one on which the methods will be performing calculations.
- *Printing maps on dynamic layers and buffering features on dynamic layers*. Features that have not been downloaded onto the displayed map will not appear in the printout or the buffer.

Additionally, the following methods do not work when the autoRefresh flag is disabled: zoomGotoDlg, zoomGotoLocation, setUrl, refresh, and setAutoRefresh. If you call one of these methods when the autoRefresh flag is disabled, they will fail and will set the Refresh Disabled error code (-14). Therefore, you need to avoid calling these methods when you have disabled the autoRefresh flag, and also avoid calling them from event-handling code for onViewChanging and onMapLoaded, as these events always disable the autoRefresh flag while running the event-handling code.

Detecting Display Refreshes

The Autodesk MapGuide Viewer fires the onViewChanging and onViewChanged events both when a display refresh is about to happen and when one just happened. You can write event-handling code in your application to respond to these events (see "Working with Autodesk MapGuide Viewer Events" on page 18). However, before the Viewer fires these events, it disables the autoRefresh flag. When writing your event-handling code for onViewChanging, be sure to avoid methods that don't work when the autoRefresh flag is disabled, as described in the previous section.

Detecting a Change in the Busy State

The Autodesk MapGuide Viewer fires the onBusyStateChanged event when the busy state changes. You can write event-handling code for this event to enable and disable specific user interface elements, such as buttons, in your application.

Handling Errors

Every application should track and handle errors in the code. The Viewer API has error tracking classes and methods that you can use while debugging your applications.

Every time an API method is run or a property is accessed, Autodesk MapGuide updates the MGError object. This object contains error information for the most recently executed method or property. You can check the error status of the most recently called method by calling the getLastError() method in MGMap. The getLastError() method returns a reference to the MGError object.

Argument Checking

If you call an API method with incorrect argument types, by default, Autodesk MapGuide Viewer has the method do nothing and flags the error in MGError. You can see which argument was incorrect by calling the getArg() method of MGError. To see the correct argument types for a method, refer to the *Autodesk MapGuide Viewer API Help*.

Debugging an Application

In addition to checking MGError, you can call the enableApiExceptions() and disableApiExceptions() methods in MGMap to throw or not throw exceptions. When exceptions are enabled and the MGError code is set to a non-zero value, Autodesk MapGuide throws an exception. Depending on your development environment, the exception will halt your code and send an error message containing the line number of the error to the screen.

Accessing Secure Data

Map authors can control whether developers can use the getVertices() and getLayerSetup() methods to access coordinate values and/or map layer setup data. Map authors control the security of this data from the Map Layer Properties dialog box in Autodesk MapGuide Author. If map authors allow access to the API, they can also stipulate that the application must send in a specific passkey first. If you are building an application for a map that requires a passkey to access the coordinate values and/or the layer setup data, you will need to get the passkey from the map author and pass it in with the unLock() method to enable the getVertices() and getLayerSetup() methods. Remember that users can view any embedded scripts in HTML, so in some cases you may

not want to hard code your passkey in your web page. To keep the passkey secure, we recommend that you implement one of the following techniques:

- Create an application that includes one frame that displays the map only. Be sure that the map fills up the entire frame. In this case, users will not be able to view the source code of the frame that displays the map. You can then hard code the passkey in the source code of that frame.
- Write a Java applet that makes a request for the passkey to your Autodesk MapGuide Server and then returns the passkey to the script in the web page. Call this applet in your embedded script after making sure that the user has met your security criteria.
- Write your entire Autodesk MapGuide Viewer application in a Java applet.

This chapter has covered the basic, essential tasks and concepts you need to understand to create your application. The next chapter discusses some of the common tasks you will perform with the Viewer API.

Viewer API Examples

In This Chapter

3

- Performing common tasks with the API
- Advanced applications

This chapter shows you how to write code for common tasks you will perform with the Viewer API. It also shows two advanced applications from beginning to end.

Performing Common Tasks with the API

This section contains some simple JavaScript code samples that show you how to perform basic Autodesk MapGuide Viewer tasks in your custom application. Note that this section refers to JavaScript code modules as *functions*, reserving the term *method* for members of the object-oriented Viewer API. Also note that although spatial data on the map is called map features, methods and properties in the Viewer API that work with map features use the term "object" instead of "feature". This difference in terminology exists because map features were called map objects in previous releases of Autodesk MapGuide. Be careful not confuse the term "object" in these API names with the object-oriented programming concept of objects. For example, the addObject method adds a map *feature* to the selection. Likewise, the MGMapObject object represents map features.

Tip The *Autodesk MapGuide Viewer API Help* contains the source code for these examples in the Examples section. For detailed information about each of the objects, methods, properties, and events, you can look them up in the index of the help.

Counting Layers

The countLayers() function counts the layers in a map and displays the count in a dialog box:

```
function countLayers()
{
   var map = getThisMap();
   var layers = map.getMapLayersEx();
   var cnt = layers.size();
   alert("This map has " + cnt + " layer(s).");
}
```

The function starts by calling getThisMap() and assigning its return value to a variable called "map":

```
var map = getThisMap();
```

Remember that getThisMap() is a custom function that detects the user's browser type and returns an MGMap object using the syntax required by that browser (see page 15).

Next, countLayers() calls getMapLayersEx(), a Viewer API method that returns an MGCollection object containing all the layers defined in the map. The layer collection is assigned to a variable called "layers":

```
var layers = map.getMapLayersEx();
```

Then it calls the MGCollection size() method, which returns a count of the layers in the collection; that number is assigned to a variable called "cnt":

```
var cnt = layers.size();
```

Finally, countLayers() displays the count, using the JavaScript alert() function:

```
alert("This map has " + cnt + "layer(s).");
```

Displaying the layer count

Listing Layers

The listLayers() function counts the layers in a map and displays their names:

```
function listLayers()
{
   var map = getThisMap();
   var layers = map.getMapLayersEx();
   var cnt = layers.size();
   var msg;
   var i;
   for (i = 0; i < cnt; i++)
   {
      var layer = layers.item(i);
      msg = msg + layer.getName() + "\n";
   }
   alert(msg);
}
```

The function starts by getting an instance of the map, a layer collection, and a layer count, using the same code as the previous example:

```
var map = getThisMap();                  // get an MGMap object
var layers = map.getMapLayersEx();       // create layer collection
var cnt = layers.size();                 // get layer count
```

Next, listLayers() uses a for loop to cycle through the layer collection, placing all the layer names in a single variable called "msg".

```
var msg;                    // empty variable to hold layer names
var i;                      // counter variable; used by loop
for (i = 0; i < cnt; i++)     // iterate from 0 to cnt, our layer count
{
   var layer = layers.item(i);           // get next layer
   msg += layer.getName() + "\n";        // add layer name to msg
}
```

The cnt variable tells the for loop to iterate one time for each map layer. At each iteration, the loop counter variable (i) is incremented and the following statements are processed.

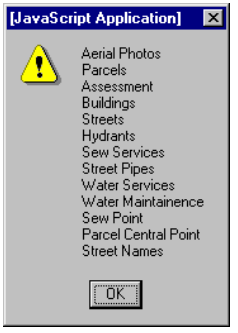
```
var layer = layers.item(i);            // get next layer
msg += layer.getName() + "\n";         // add layer name to msg
```

The first statement uses the item() API method to select a layer from the collection and assign it to a variable called “layer”.

The second statement operates on the layer variable, first using the API method getName() to obtain the name of the layer represented by that variable, and then assigning that name to the msg variable. In addition to the layer name, msg is also assigned its previous contents and the JavaScript newline character, \n. This has the effect of adding each layer name to msg as a separate text line.

Finally, listLayers() uses the JavaScript alert() function to display the contents of msg in a dialog box:

```
alert(msg);
```



Displaying layer names

Adding a Layer

The doAddLayer() function adds a single named layer to a map:

```
function doAddLayer()
{
   if (navigator.appName == "Netscape")
      document.myMap.addMapLayer("hydro.mlf", document.obs);
   else
      window.myMap.addMapLayer("hydro.mlf");
}
```

The function starts by checking the browser type. If the browser is Netscape, doAddLayer() calls the addMapLayer() API method, supplying it with the

name of an existing map layer file (MLF) and the name of the Java observation applet:

```
document.myMap.addMapLayer("hydro.mlf", document.obs);
```

If the browser is Internet Explorer (or, more specifically, not Netscape), doAddLayer() calls the addMapLayer() API method, supplying only the layer name as an argument:

```
window.myMap.addMapLayer("hydro.mlf");
```

Because addMapLayer() takes different arguments depending on the browser type, we didn't bother to return the map object with getThisMap(). Instead, we supplied the map object using the syntax required by each browser.

Note If you are supporting the Netscape browser, you must provide the name of the Java observation applet as a second argument to addMapLayer().

Linking Layers

Autodesk MapGuide Author enables you to set map layer attribute properties for specific display ranges. (Refer to "Setting Style Properties for Layers" in the *Autodesk MapGuide User's Guide*.) For example, you might set a layer to be invisible when a user zooms out. Using the Viewer API, you can extend this functionality by linking layers to one or more designated control layers. Then, if an action such as a zoom-out causes a control layer to become invisible, the API can make the linked layers invisible as well.

In the following example, the onViewChanging() function checks the visibility of three control layers named "States", "Counties", and "ZIP Codes". If one or more of the control layers is visible, onViewChanging() makes all other map layers visible; if none of the control layers is visible, onViewChanging() suppresses visibility of all other map layers.

We've named our function onViewChanging() because it's triggered by the Viewer API event of the same name. Whenever a Viewer API event is triggered, the Viewer checks for a function whose name matches the event name. If the function is found, the Viewer invokes it, passing arguments that vary by event.

Unlike the previous examples, onViewChanging() takes an argument, which is an MGMap object passed by the onViewChanging event. Because the event provides an instance of the map object, we don't need to obtain it with getThisMap().

```
function onViewChanging(thisMap)  // 'thisMap' is MGMap object provided by event
{
   var states = thisMap.getMapLayer ("States");
   var countries = thisMap.getMapLayer("Counties");
   var zipCodes = thisMap.getMapLayer("ZIP Codes");
   var vis =
      (    states.getVisibility() ||
           counties.getVisibility() ||
           zipCodes.getVisibility()      );
   var layers = thisMap.getMapLayersEx();
   for (var i = 0; i < layers.size(); i++)
   {
      var layer = layers.item(i);
      if (!layer.equals(states)
         && !layer.equals(counties) && !layer.equals(zipCodes))
      {
         layer.setVisibility(vis);
      }
   }
}
```

The function starts by using the getMapLayer() API method to return each of the control layers as objects. Those objects are assigned to three variables named "states", "counties", and "zipCodes":

```
var states = thisMap.getMapLayer ("States");
var countries = thisMap.getMapLayer("Counties");
var zipCodes = thisMap.getMapLayer("Zip Codes");
```

Next, onViewChanging() uses the getVisibility() method to determine if any of the control layers are visible. If at least one control layer is visible (i.e., if states is visible *or* counties is visible *or* zipCodes is visible), getVisility() returns the Boolean value "true", thus setting the vis variable to "true". Otherwise, it sets vis to "false":

```
var vis =
  ( states.getVisibility() || counties.getVisibility() || zipCodes.getVisibility() );
```

Then onViewChanging() uses the getMapLayersEx() method to create a layer collection and assign it to a variable named "layers":

```
var layers = thisMap.getMapLayersEx();    // create layer collection
```

Finally, the function uses a for loop to cycle through each map layer. Each time the loop encounters a layer that is not one of the control layers, that layer is made visible or invisible, depending on the value of vis:

```
for (var i=0; i < layers.size(); i++)
{
   var layer = layers.item(i);
   if (!layer.equals(states) && !layer.equals(counties)
      && !layer.equals(zipCodes))
   {
      layer.setVisibility(vis);
   }
}
```

Retrieving Keys of Selected Features

In this example, the doGetKey() function displays a dialog box showing the keys of selected map features (keys are unique values that are used to identify individual map features). If no features are selected, an alert displays prompting the user to make a selection:

```
function doGetKey()
{
   var map = getThisMap();
   if (map.getSelection().getNumObjects() == 0)
   {
      alert ("Please make a selection first.");
      return;
   }
   var sel = map.getSelection();
   var objs = sel.getMapObjectsEx(null);
   var cntObjects = objs.size();
   var msg = "Keys of selected features are:\n";
   var i;
   for (i = 0; i < cntObjects; i++)
   {
      var obj = objs.item(i);
      var key = obj.getKey();
      msg = msg + obj.getMapLayer().getName() + " " + key + "\n";
   }
   alert(msg);
}
```

The function starts by getting an instance of MGMap:

```
var map = getThisMap();
```

Then it uses two API methods to see if any features are selected. Note that the methods are concatenated; the first method, getSelection(), operates on the map and returns a selection object, which is then passed to the second method, getNumObjects(), for processing. If no features are selected, an alert displays and the function terminates; otherwise, the selection is assigned to a variable named "sel":

```
if (map.getSelection().getNumObjects() == 0)
{
   alert ("Please make a selection first.");
   return;
}
var sel = map.getSelection();
```

Next, doGetKey() calls the getMapObjectsEx() API method and passes its return value (a collection of the selected features) to a variable called "objs". Note that if you use getMapObjectsEx() with a map layer, it returns an MGCollection object made up of all the features on a layer, but by using the

method with the selection object, and by passing it "null" as a parameter, it returns the features in the current selection only:

```
var objs = sel.getMapObjectsEx(null);
```

Then the function calls the MGCollection size() method, which returns a count of the objects in the collection; that number is assigned to a variable called "cntObjects":

```
var cntObjects = objs.size();
```

After that, doGetKey() uses a for loop to cycle through the feature collection, placing all of the feature names in a single variable called "msg":

```
var msg = "Keys of selected features are:\n"; // variable to hold feature names
var i;                                        // loop counter variable
for (i = 0; i < cntObjects; i++)              // iterate from 0 to cntObjects
{
   var obj = objs.item(i);
   var key = obj.getKey();
   msg = msg + obj.getMapLayer().getName() + " " + key + "\n";
}
```

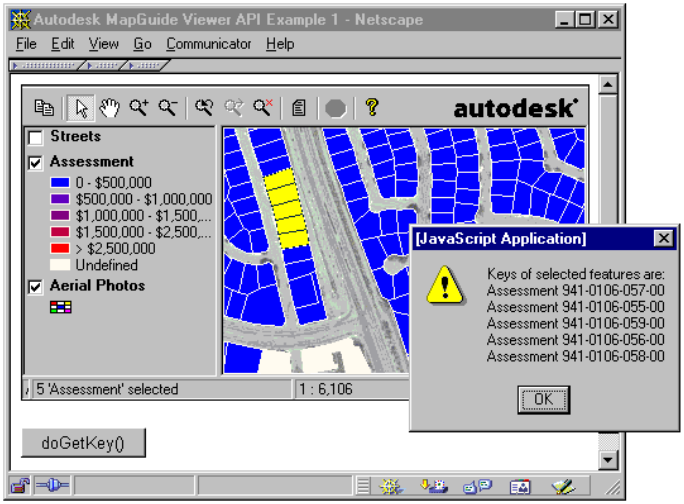
The cntObjects variable tells the for loop to iterate one time for each object. At each iteration, the loop counter variable (i) is incremented and the following statements are processed:

```
var obj = objs.item(i);
var key = obj.getKey();
msg = msg + obj.getMapLayer().getName() + " " + key + "\n";
```

The first statement uses the item() API method to select an object from the collection and assign it to a variable called "obj".

The second statement operates on the obj variable, first using the getKey() API method to obtain the key of the feature represented by that variable, and then assigning that name to the msg variable. The last line puts it all together by concatenating the previous contents of msg, a layer name obtained by the getName() method, a space character, the contents of key, and a JavaScript newline. After all selected features have been added to msg, the contents of the variable are displayed in a JavaScript alert() box:

```
alert(msg);
```



Displaying keys of selected features

Retrieving Coordinates of a Selected Feature

The doGetCoordinates() function displays a dialog box showing the coordinates of a selected feature:

```
function doGetCoordinates()
{
   var map = getThisMap();
   var sel = map.getSelection();
   var layer = map.getMapLayer("Parcels");
   if (layer == null)
   {
      alert("No Parcels layer found in this map.");
      return;
   }
   if ( (sel.getNumObjects() > 1) || (sel.getNumObjects() == 0) ||
        (sel.getMapObjectsEx(layer).size() == 0) )
   {
      alert("Select only one parcel, please.");
      return;
   }
   var obj = sel.getMapObjectsEx(layer).item(0);
   var vertices = map.createObject("MGCollection");
   var cntVertices = map.createObject("MGCollection");
   var res = obj.getVertices(vertices, cntVertices);
   if (res == 0)
   {
      alert("No access to coordinate information.");
      return;
   }
   msg = "Parcel:" + obj.getKey() + "\n";
```

```
   msg = msg + "Coordinates in MCS unit\n";
   for(var i = 0; i < cntVertices.item(0); i++)
   {
      var pnt = vertices.item(i);
      msg = msg + pnt.getX() + "," + pnt.getY() + "\n";
   }
   alert(msg);
}
```

The function starts by using getThisMap() to get an instance of the map; then it gets the current selection and assigns it to a variable called "sel":

```
var map = getThisMap();
var sel = map.getSelection();
```

Then doGetCoordinates() uses the getMapLayer() method to select the "Parcels" layer and assign it to a variable named "layer"; if the layer doesn't exist in the map, an alert displays and the function terminates:

```
   var layer = map.getMapLayer("Parcels");
   if (layer == null)
   {
      alert("No Parcels layer found in this map.");
      return;
   }
```

Next, doGetCoordinates() uses the getNumObjects() and getMapObjectsEx() methods to verify that one, and only one, feature is selected, and that the current layer is not empty. If the criteria are not met, an alert displays and the function terminates:

```
if (     (sel.getNumObjects() > 1) ||
         (sel.getNumObjects() == 0) ||
         (sel.getMapObjectsEx(layer).size() == 0)     )
{
   alert("Select only one parcel, please.");
   return;
}
```

After that, the function creates some more variables. The "obj" variable contains the first (and only) object in the current selection. The "vertices" and "cntVertices" variables hold empty MGCollection objects:

```
var obj = sel.getMapObjectsEx(layer).item(0);
var vertices = map.createObject("MGCollection");
var cntVertices = map.createObject("MGCollection");
```

Then doGetCoordinates() uses the getVertices() method to get the coordinates and number of vertices of obj, our selected parcel. The values getVertices() obtains are passed to the empty vertices and cntVertices collections.

If getVertices() is successful, it returns an integer telling the number of vertices it found; otherwise, it returns zero. The getVertices() return value is

passed to a variable called “res”. If getVertices() returns zero, an alert displays and the function terminates:

```
var res = obj.getVertices(vertices, cntVertices);
if (res == 0)
{
   alert("No access to coordinate information.");
   return;
}
```

Next, doGetCoordinates() uses a for loop to cycle through the vertices collection, placing all of the coordinate listings in a single variable called “msg”:

```
msg = "Parcel:" + obj.getKey() + "\n";
msg = msg + "Coordinates in MCS unit\n";
for(var i = 0; i < cntVertices.item(0); i++)
{
   var pnt = vertices.item(i);
   msg = msg + pnt.getX() + "," + pnt.getY() + "\n";
}
```

The cntVertices variable tells the for loop to iterate one time for each vertex in the object. At each iteration, the loop counter variable (i) is incremented and the following statements are processed:

```
var pnt = vertices.item(i);
msg = msg + pnt.getX() + "," + pnt.getY() + "\n";
```

The first statement uses the item() API method to select a vertex from the collection and assign it to a variable called “pnt”.

The second statement operates on the pnt variable, using the getX() and getY() methods to get the vertex coordinates and assign them to msg. As with the previous examples, a new line is added to msg each time the for loop iterates. After all coordinates have been added to msg, the contents of the variable are displayed in a JavaScript alert() box:

```
alert(msg);
```

Invoking Select Radius Mode

The doSelectRadius() function gets an instance of the map and uses that instance to call the selectRadiusMode() API method:

```
function doSelectRadius()
{
   var map = getThisMap();
   map.selectRadiusMode();
}
```

Radius Mode allows the user to digitize a circle and select all map features that fall within that circle.

Toggling a Layer On and Off

The layerToggle() function toggles the visibility of a layer that is specified when the function is invoked:

```
function layerToggle(l_name)
{
   var map = getThisMap();
   var layer = map.getMapLayer(l_name);
   if (layer == null)
      alert("layer not found.");
   else
   {
      layer.setVisibility(!layer.getVisibility());
      map.refresh();
   }
}
```

This function takes a layer name as an argument. For instance, it might be called as follows:

```
<FORM>
<INPUT TYPE="button" VALUE="Toggle Hydro" ONCLICK="layerToggle('Hydro')">
</FORM>
```

The layerToggle() function starts by getting an instance of the map object. Then it passes the function argument, l_name, to the getMapLayer() API method. The getMapLayer() function returns the specified layer, or it returns "null" if the layer is not found. The getMapLayer() return value is then assigned to a variable named "layer":

```
var map = getThisMap();
var layer = map.getMapLayer(l_name);
```

Next, the function checks the value of layer. If it is null, an alert displays; otherwise the setVisibility() method is used to toggle the layer's visibility to the opposite of its current state:

```
if (layer == null)
   alert("layer not found.");
else
{
   layer.setVisibility(!layer.getVisibility());
   map.refresh();          // after changing visibility, refresh map
}
```

Note the use of the not operator (!) with the getVisibility() method. This has the effect of checking the layer's visibility and returning the opposite of what it finds.

Zooming In on Selected Features

The zoomSelect() function zooms in to a selected feature:

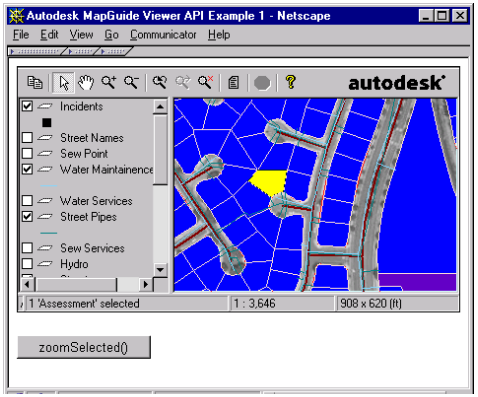
```
function zoomSelect()
{
   var map = getThisMap();
   var selected = map.getSelection().getMapObjectsEx(null);
   if (selected.size()>0)
      map.zoomSelected();
   else
      alert("Nothing selected.");
}
```

First, the function gets an instance of MGMap. Then, it uses two concatenated API methods to retrieve selected features and pass them to the variable "selected". The first method, getSelection(), returns a selection object, which is used by the second method, getMapObjectsEx(). If you use getMapObjectsEx() with a map layer, it returns an MGCollection object containing all features on the layer, but by using getMapObjectsEx() with the selection object and passing it "null", it returns the features in the current selection only:

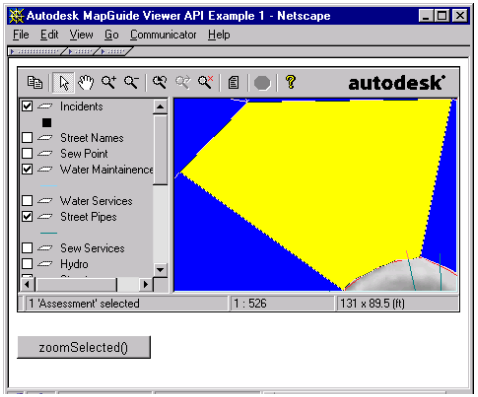
```
var map = getThisMap();
var selected = map.getSelection().getMapObjectsEx(null);
```

Next, zoomSelect() uses the size() method to see how many features are selected. If one or more features are selected, the zoomSelected() API method is invoked, causing the Viewer to zoom to those features. Otherwise, an alert displays and no zoom occurs:

```
if (selected.size() > 0)
   map.zoomSelected();
else
   alert("Nothing selected.");
```



Before calling zoomSelected()



After calling zoomSelected()

Counting Map Features

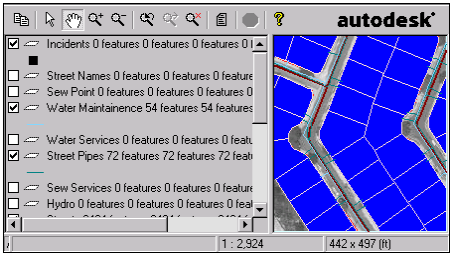
The showFeatureCount() function counts the features on each map layer and adds that count to the legend:

```
var legendSet;       // Global variable, declared outside of function

function showFeatureCount()
{
   if (legendSet)
      return;
   var map = getThisMap();
   if (map.isBusy() == false)
   // can also be written as 'if (!map.isBusy())'
   {
      var layers = map.getMapLayersEx();
      var cnt = layers.size();
      var i;
      var msg;
      for (i = 0; i<cnt; i++)
      {
      var layer = layers.item(i);
      var objectCount = layer.getMapObjectsEx().size();
      var label = layer.getLegendLabel();
      label = label + " " + objectCount + " features";
      layer.setLegendLabel(label);
      }
   }
   legendSet = true;
}
```

The function starts by checking the status of the global variable, legendSet. If legendSet is set to “true”, showFeatureCount() terminates:

```
if (legendSet)
   return;
```

This keeps showFeatureCount() from printing multiple messages to the legend if the user clicks the button more than once, as illustrated below:



Next, showFeatureCount() creates an instance of the map and checks to see if the map is in a busy state:

```
var map = getThisMap();
if (map.isBusy() == false)
```

If the map is not busy, the function continues.

First, it uses the getMapLayersEx() method to obtain a layer collection and assign it to a variable called "layers". Then it uses the size() method to get the number of layers and assign that number to a variable called "cnt":

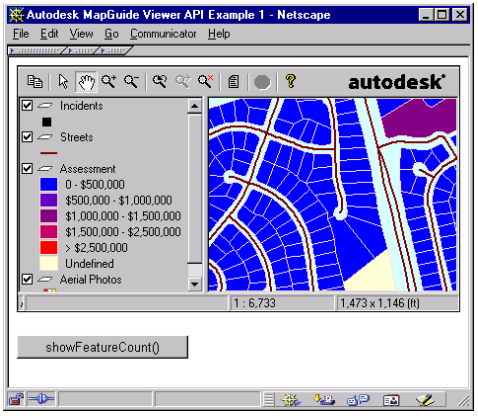
```
   var layers = map.getMapLayersEx();
   var cnt = layers.size();
```

Then it creates a loop that counts the features in each layer and uses the getLegendLabel() and setLegendLabel() methods to report the count in the map legend:

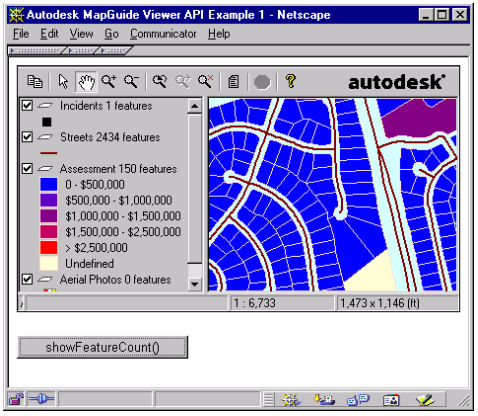
```
var i;
var msg;
for (i = 0; i<cnt; i++)
{
   var layer = layers.item(i);
   var objectCount = layer.getMapObjectsEx().size();
   var label = layer.getLegendLabel();
   label = label + " " + objectCount + " features";
   layer.setLegendLabel(label);
}
```

Finally showFeatureCount() sets the global legendSet variable to "true". This keeps the function from running again until the page containing the map is refreshed.

```
legendSet = true;
```



Before calling showFeatureCount()



After calling showFeatureCount()

Customizing the Printout

Autodesk MapGuide lets map authors and Viewer users control how the printed map appears on a page. For example, a map author might create a custom symbol that displays only in the printout. Or, when printing from the Viewer, a Viewer user might choose to change the map title or suppress page elements such as the legend, scale bar, or north arrow. The API supports these user-interface features and also provides additional functionality, allowing you to write code to change the title font, add a custom symbol, or control the size and position of any page element on the printout.

As a developer, you can specify that two events be fired each time a map is sent to the printer. The first event, onBeginLayout, is fired after a user clicks OK in the Print dialog box, but before the Viewer lays out the page elements that will be sent to the printer. The second event, onEndLayout, is called after the Viewer lays out the page elements, but before the elements are sent to the printer. By writing event handler functions for these events, you can intercept the page before it gets to the printer and customize it to your liking.

Enabling the Print Events

By default, onBeginLayout and onEndLayout are not fired; you enable and disable them using the enablePrintingEvents and disablePrintingEvents methods. For Netscape and the Java edition, you will also need to use the setPrintingObserver method to specify the event observer. Here's one way to write a JavaScript function that enables print events:

```
function enable_print_events()
{
    var map = getThisMap();
    map.enablePrintingEvents();
    if (navigator.appName == "Netscape")
        map.setPrintingObserver(obs);
}
```

Writing Event Handler Functions

The sections that follow show how to write JavaScript and Java event handlers for onBeginLayout and onEndLayout.

onBeginLayout

If you want your event handling code to control settings from the Viewer's Page Setup dialog box, you should attach it to the onBeginLayout event. Two objects, an MGPageSetup and an MGPrintInfo, are passed automatically to onBeginLayout when that event is fired:

```
void onBeginLayout (MGPageSetup pgSetup, MGPrintInfo info)
```

The MGPageSetup object describes the state of the Page Setup dialog box immediately before the user clicked OK in the Print dialog box. The MGPrintInfo object provides information about the resolution of the output device and the size of the printable area of the page.

The following example shows one way to write an onBeginLayout event handler that suppresses all page elements except the map. The example assumes you've set up the Java event applet, and that you've already enabled the print events, as shown above.

First, create a button on the HTML page:

```
<form>
    <input type="button" value="Just the Map"
        OnClick="print_map_only();" name="myButton">
</form>
```

Then, create a JavaScript function that the button will call. In this example, the function sets the state of a boolean variable called "map_only". The variable will be read by our event handler, so we've given it global scope by declaring it outside the function body.

```
var map_only;  // put var outside function body
function print_map_only()
{
    map_only = "true";
    getThisMap().printDlg();
    map_only = "false";
}
```

Finally, we write our event handler. It goes in the HTML page (or *.js* file), just like any other JavaScript function. This function is executed automatically every time the onBeginLayout event fires. Note that the function takes an MGPageSetup and an MGPrintInfo as its parameters.

```
function onBeginLayout (pgSetup, info)
{
    if (map_only == "true")
    {
        pgSetup.setInclude("mg_legend", false);
        pgSetup.setInclude("mg_northarrow", false);
        pgSetup.setInclude("mg_scalebar", false);
        pgSetup.setInclude("mg_title", false);
        pgSetup.setInclude("mg_timestamp", false);
        pgSetup.setInclude("mg_url", false);
    }
}
```

Note You can control the Page Setup without using onBeginLayout, but the results are different. In the example above, the Page Setup is modified only for that printout. Because the event handler is working with a copy of the MGPageSetup object, subsequent printouts from the popup menu don't show these changes, and the changes don't appear in the Page Setup dialog box. If you

were to write a similar function that was not attached to the onBeginLayout event, the changes would continue to be reflected in both the printout and the Page Setup dialog box until the map is refreshed.

onEndLayout

If you want your event handling code to change the title font, add a custom symbol, or control the position and size of any page element, you should attach it to the onEndLayout event. Two objects, an MGPrintLayout and an MGPrintInfo, are passed automatically to onEndLayout when that event is fired:

```
void onEndLayout (MGPrintLayout prLayout, MGPrintInfo info)
```

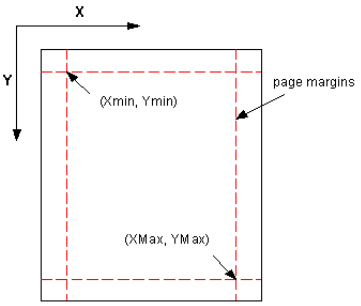
The MGPrintLayout object provides access to printed page elements; you can then use MGPageElement and MGExtentEx to control how those elements display. The MGPrintInfo object provides information about the resolution of the output device and the size of the printable area of the page.

The following example shows a printing event handler, written in Java, that adds a custom symbol to the printout. Note that this example includes event handlers for both onEndLayout and onBeginLayout. The example assumes you've set up the Java event applet, and that you've already enabled the print events:

```
public class MyObserver extends Applet implements MGPrintingObserver
{
    public void onBeginLayout(MGPageSetup pgSetup, MGPrintInfo info)
     {
         // turn off all elements except the map
         pgSetup.setInclude("mg_scalebar", false);
         pgSetup.setInclude("mg_northarrow", false);
         pgSetup.setInclude("mg_title", false);
         pgSetup.setInclude("mg_timestamp", false);
         pgSetup.setInclude("mg_legend", false);
     }

     public void onEndLayout(MGPrintLayout layout, MGPrintInfo info)
     {
         int pixelsPerInch = info.getPageResolution();

         // retrieve the page elements from the MGPrintInfo class
         MGPageElement mapEle = layout.getPageElement("mg_map");
         MGPageElement logoEle = layout.addSymbol("mylogo");

         // get the extents of the page elements
         MGExtentEx mapExt = mapEle.getExtent()
         MGExtentEx logoExt = logoEle.getExtent();

         // set the width and height of the logo to 1" by 1"
         logoExt.set(mapExt.getMinX(), mapExt.getMinY(),
             mapExt.getMinX() + pixelsPerInch,
             mapExt.getMinY() + pixelsPerInch);
        logoEle.setExtent(logoExt);
     }
}
```

Positioning Page Elements with the Print Coordinate System

Page element extents specified through the API are expressed in Page Coordinate System (PCS) units. The origin (0,0) of this system is located at the upper-left corner of the paper. Its exact location depends on the current left and top margins. Unlike the coordinate system that is used on a map, the Y values increase in the downward direction and the X values increase to the right. Like the device unit type, the default PCS unit type is a pixel.



Setting the Print Priority

As shown above, you can write an onEndLayout event handler that uses MGPrintLayout, MGPageElement, and MGExtentEx to control the placement of printed page elements. It is possible, and sometimes desirable, to place page elements on top of each other. For example, you might want to move the north arrow on top of an empty spot of ocean in your map. Of course, this doesn't do your user much good if the ocean prints on top of the north arrow and hides it.

To solve this problem, each page element is assigned a default print priority. A print priority is a positive floating-point number between 0.0 and 100.0 that describes the relative printing order of a page element. The element with the lowest number is printed first. The element with the highest number is printed last. You can read and change an element's priority with the getPrintPriority and setPrintPriority methods. The default print priority values are as follows:

Element	Default Print Priority
map	10.0
legend	20.0
title	30.0
URL	40.0
date/time	50.0
scale bar	60.0
north arrow	70.0
custom elements	80.0

The following example shows an onEndLayout event handler, written in JavaScript, that forces the title to be printed after the north arrow.

```
function onEndLayout(layout, info)
{
    // retrieve arrow and map elements
    var el_arrow = layout.getPageElement("mg_northarrow");
    var el_map = layout.getPageElement("mg_map");

    // force arrow to have higher print priority than map
    el_arrow.setPrintPriority(el_map.getPrintPriority() + 1);
}
```

Adding Custom Page Elements

You can add custom page elements to the printout. Currently, the API can only access symbols in the API symbol list. The API symbol list is a WMF or EMF file containing a small set of predefined symbols. Additional symbols can be added to the list using Autodesk MapGuide Author. Refer to the *Autodesk MapGuide Help* for more information.

The following example shows an onEndLayout event handler, written in JavaScript, that adds a custom logo to the top left corner of the printout. Note that the logo is rotated 90°. This example assumes that the "MyLogo" symbol has been added to the API symbol list by the map author:

```
function onEndLayout(layout, info)
{
    // add 'MyLogo' symbol to layout and return as 'sym'
    var sym = layout.addSymbol("MyLogo");

    // function ends if symbol doesn't load properly
    if (sym == null) return;
// display symbol the top-left corner of page
    MGExtentEx ext = sym.getExtent();
    ext.set(0, 0, 600, 600);
    sym.setExtent(ext);

    // rotate symbol
    var attr = sym.getSymbolAttr();
    if (attr != null) {
        attr.setRotation(-90.0);
    }
}
```

Advanced Applications

Now that you know how to create a basic application and perform common tasks, you are ready to explore the more advanced applications you can create with the Viewer API. This section provides three examples of how you might use the Viewer API to create an advanced application. The Autodesk MapGuide web site provides several examples, both demo applications and real customer sites, that demonstrate how to create solutions for complex needs. You can find the links to the demo applications, customer sites, and more at *www.mapguide.com*.

When you have finished reviewing the following example, be sure to read Chapter 4, "Using Reports to Query and Update Data Sources." Chapter 4 provides more information about creating custom reports and server-side scripts that enable users to dynamically update attribute data sources, a technique illustrated in the municipal demo, which begins on page 53.

Custom Redlining Application

Redlining applications allow a user to add annotations to a drawing or map without using the original authoring application or modifying the original document. You can use the Autodesk MapGuide Viewer API to create a custom redlining application that allows users to mark up a map using the Viewer. The user's markups are saved to a special layer type called a client redline layer. Client markups can be printed or saved (along with the rest of the map) to an MWF.

You can make your application as sophisticated as your needs warrant, but the basic process for creating a redlining application is simple:

1 Use the createLayer method to create the redline layer, or, if the layer already exists, you can access it with getMapLayer.

2 Use createMapObject to add an empty redline object to the layer.

3 Use the MGMapObject add methods to add one or more primitives to the redline object (primitives are the individual symbols, polylines, polygons, or text blocks that make up a redline object). You can add a single primitive or combine several primitives to create a complex object. For example, you might create a complex object consisting of an arrow and a text callout.

4 Use the saveMWF method to save the map file to an MWF on the user's machine or a network server.

Note For information about adding and deleting features from the data source itself (such as an SDF file), rather than saving changes to an MWF, see "Updating SDF Files via the Map" on page 136.

A Redlining Application Example

The following example shows one way to write a simple redlining application. The application lets the user create a redlining layer, add polygon objects to that layer, and save the map to a drive on a local machine. The user interface is sparse, consisting of a small HTML form with a text box and two buttons, as shown in the following illustration.

Redlining example

Here is the code for the form:

```
<FORM NAME="the_form">
    Polygon Name: <INPUT TYPE="text" VALUE="" NAME="the_textbox">
    <INPUT TYPE="button" VALUE="Add/Update Polygon" On-
Click="add_pgon();" NAME="a_button">
    <INPUT TYPE="button" VALUE="Save It!" OnClick="save_it();"
NAME="another_button">
</FORM>
```

The Add/Update Polygon button calls a JavaScript function that lets the user draw a polygon by digitizing points on the map. The function first checks to see if there's a value in the Polygon Name check box. If there is a value, the function calls either the digitizePolygon or digitizePolygonEx method. Otherwise, the function displays an alert and exits:

```
function add_pgon()
{
    // get map object
    var map = getThisMap();

    // exit function if 'Polygon Name' text box is empty
    if (document.the_form.the_textbox.value == "")
    {
        alert("Please enter a polygon name.")
        return;
    }

    // if browser is Netscape, use 'Ex' version and pass
    // observer applet; if browser is Internet Explorer,
    // use 'non-Ex' version with no argument
    if (navigator.appName == "Netscape")
        map.digitizePolygonEx(document.obs);
    else
        map.digitizePolygon();
}
```

The digitizePolygon and digitizePolygonEx methods both fire the onDigitized-Polygon event, passing it the map object, the number of polygon vertices, and the coordinates of those vertices. The onDigitizedPolygon event looks for a JavaScript function of the same name and, if that function exists, executes it. In fact, the onDigitizedPolygon function does exist, because we've created it. Here's the code for that function:

```
function onDigitizedPolygon(map, numPoints, points)
{
    // create variable and assign it user-specified value
    // from 'Polygon Name' text box
    var formText = document.the_form.the_textbox.value;

    // create redline layer, or get it if it already exists
    var myLayer = map.getMapLayer("My Redline Layer");
    if (myLayer == null)
        myLayer = map.createLayer("redline", "My Redline Layer");
```

```
    // create redline object or get it if it exists (getMapObject
    // takes an object key as its value, while createMapObject takes
    // a key and a name -- the formText variable supplies both of
    // those values)
    var obj = myLayer.getMapObject(formText);
    if (obj == null)
        var obj = myLayer.createMapObject(formText, formText, "");

   // create MGCollection that holds user-specified polygon vertices
    var user_vertices = map.createObject("mgcollection");
    user_vertices.add(numPoints);

    // use MGCollection to create polyline primitive and add it to
    // redline object
    obj.addPolylinePrimitive(points, user_vertices, false);

    // clear contents of 'Polygon Name' text box
    document.the_form.the_textbox.value = "";
}
```

The Save It! button calls a JavaScript function that saves the map to the user's hard drive. The function prompts the user for the map password, then calls the saveMWF method and saves the map to the specified path:

Note
Be sure to use "/" or "\\" in the path, not "\".

```
function save_it()
{
var fName = "c:\\My Documents\\my_map.mwf";
var password = prompt("Please enter a password.", "");

    if ( getThisMap().saveMWF(fName, password) )
        alert("Map has been saved!");
    else
        alert("Unable to save map.");
}
```

For More Information

This example was a very simple application designed to illustrate redlining, but you will probably want your application to have more features, such as allowing users to add other primitives besides polygons. You might also want it to exert more control over how the primitives appear onscreen or to query the state of existing redline objects. To learn more about these topics, refer to the following sections in the *Autodesk MapGuide Viewer API Help*:

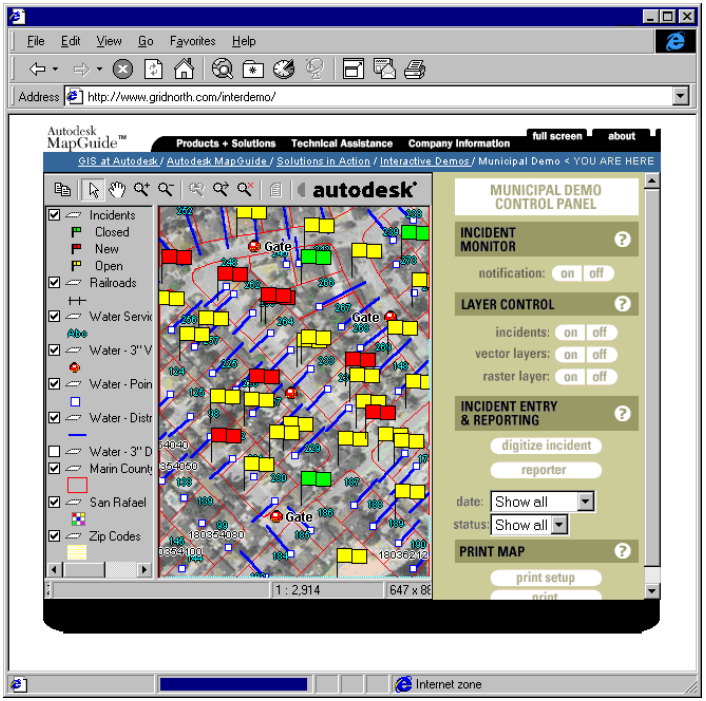
- For information about creating primitives, look up the MGMapObject add methods (addCirclePrimitive, addPolygonPrimitive, etc.). Also look up the MGMap digitize methods and their corresponding events.
- For information about controlling the appearance of redline objects, look up MGEdgeAttr, MGFillAttr, MGLineAttr, MGSymbolAttr, MGTextAttr, and MGRedlineSetup.
- For information about querying redline objects, look up MGPrimitive.

Municipal Application

This application demonstrates how you could monitor the water and sewer systems of a city. In the event of water distribution system problems, the application can notify the user, or a user can add an incident to the map and generate reports. The application uses color digital imagery to help orient the user.

On the left side of the window, the standard legend allows you to turn layers on and off and select them. On the right side of the window, notice the additional controls. These controls interact with the map through the Viewer API.

The application includes an incident monitor, which can notify the user if there are problems with the water distribution system. The layer control allows you to turn off the incident layer, turn off all vector layers at once, and turn off the raster layer. These controls are useful for finding information quickly. Lastly, the “digitize incident” and “reporter” buttons allow the user to add an incident to the map and generate reports about selected features.



Municipal application

Source Code

Following is the source code for the controls. Additional comments have been added to the code to give you a better idea of how the scripting works. To view the source code for the other frames in this application, go to the application online at *www.autodesk.com/mapguidedemo*.

Municipal Application

```
<HTML>
<HEAD>
<TITLE>MUNICIPAL</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--

// Get full browser name and assign it to tempName variable;
// then assign first 8 letters of tempName to browserName variable
var tempName = navigator.appName;
var browserName = tempName.substring(0,8);

// Map object variable, to be used later
var MyMap;

// Set browserId variable: '1' for netscape, '2' for IE, else '0'
if (browserName == 'Netscape')
  var browserId = 1;
else if (browserName == 'Microsof') // just first 8 letters...
  browserId=2;
else
  browserId = 0;

// =---------=---------=---------=---------=---------=---------=-= //
// Function: getMyMap()
// Description: Get appropriate map object for IE or Netscape
// **Same concept as getThisMap() example on page 15)**
// Arguments: none
// Returns: map object
// =---------=---------=---------=---------=---------=---------=-= //
function getMyMap()
{
  // Get appropriate MGMap object; type depends on browserId value
  if (browserId == 1)
    MyMap = top.main.document.embeds[0]; // Netscape map object
  else if (browserId == 2)
    MyMap = top.main.document.MyMap;     // IE map object
  else
    MyMap = null;                        // none if other browser
  return MyMap;                          //return map object
}
```

Municipal Application (*continued*)

```
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: notify()
// Description: Turns on Incident Monitor by calling
// CreateInWindow() function, below
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function notify()
{
  // Call function
  CreateInWindow();
}
```

For more info...
See *Chapter 4, "Using Reports to Query and Update Data Sources."*

```
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: CreateInWindow()
// Description: Calls a ColdFusion file and generates the
// resulting incident report in a new window (called by 'On'
// button, under 'Incident Monitor')
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //

// First, a global variable, defined *outside* of the function body
var InWindow;

// Function starts here
function CreateInWindow()
{
  // Set 'URL' variable to location of ColdFusion file used to
  // generate report (see Chapter 4 for more information); then
  // open URL in new window called 'InWindow'
  var URL =
    "http://www.gridnorth.com/interdemo/municipal/reports/Incident1.cfm"
  InWindow = open(URL,"InWindow", "toolbar=no,width=480,
           height=350,directories=no,status=no,
           scrollbars=YES,resizable=YES,menubar=no")
}

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: notifyoff()
// Description: Closes window containing incident report
// (called by 'Off' button, under 'Incident Monitor')
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function notifyoff()
{
  // If 'InWindow' doesn't exist, or if it's already closed,
  // terminate function; otherwise close 'InWindow'
```

Municipal Application (*continued*)

```
  if ( (InWindow == null) || (InWindow.closed) )
    return;
  else
    InWindow.close();
}

// =-=========-=========-=========-=========-=========-=========-=========-= //
// Function: Incidents(type)
// Description: Turns Incidents layer on or off (called by
// ON/OFF buttons, under Layer Control: Incidents)
// Arguments: string ('On' or 'Off')
// Returns: nothing
// =-=========-=========-=========-=========-=========-=========-=========-= //
function incidents(type)
{
  myMap = getMyMap();            // Get instance of MGMap
  if (myMap.isBusy() == false)   // If Viewer is not busy...
  {
    // ...get "Incidents" layer
    var MyLayer1 = myMap.getMapLayer("Incidents");

    // If function was called with 'Off', make the
    // layer invisible; otherwise make the layer visible
    if (type == 'Off')
      MyLayer1.setVisibility(false);
    else
      MyLayer1.setVisibility(true);

    // Tell the Viewer to rebuild layer when map is refreshed;
    // then refresh map
    MyLayer1.setRebuild(true);
    myMap.refresh();
  }
  // If Viewer is busy, don't do the stuff above;
  // instead, display alert
  else
     alert("The Viewer is busy. Please try again in a few seconds.");
}

// =-=========-=========-=========-=========-=========-=========-=========-= //
// Function: vector(type)
// Description: Turns vector layers on or off (called by
// ON/OFF buttons, under Layer Control: Vector Layers)
// Arguments: string ('On' or 'Off')
// Returns: nothing
// =-=========-=========-=========-=========-=========-=========-=========-= //
function vector(type)
{
  myMap = getMyMap();            // Get instance of MGMap
```

Municipal Application (*continued*)

```
if (myMap.isBusy() == false)   // If Viewer is not busy...
{
  // ...assign a bunch of layers to a bunch of variables
  var MyLayer2 = myMap.getMapLayer("Population > 100000");
  var MyLayer3 = myMap.getMapLayer("Population > 50000");
  var MyLayer4 = myMap.getMapLayer("Population > 500");
  var MyLayer5 = myMap.getMapLayer("Population > 0");
  var MyLayer6 = myMap.getMapLayer("Railroads");
  var MyLayer7 = myMap.getMapLayer("Interstates");
  var MyLayer8 = myMap.getMapLayer("Highways");
  var MyLayer9 = myMap.getMapLayer("Major Roads");
  var MyLayer10 = myMap.getMapLayer("Minor Roads");
  var MyLayer11 = myMap.getMapLayer("Water Service Acct.");
  var MyLayer12 = myMap.getMapLayer("Water - 3 inch Valves");
  var MyLayer13 = myMap.getMapLayer("Water - Point of Service");
  var MyLayer14 = myMap.getMapLayer("Water - Distribution");
  var MyLayer15 =
    myMap.getMapLayer("Water - 3 inch Distribution");

  var MyLayer16 =
    myMap.getMapLayer("Marin County Land Parcels");
  var MyLayer17 = myMap.getMapLayer("Population Density");
  var MyLayer18 = myMap.getMapLayer("ZIP Codes");
  var MyLayer19 = myMap.getMapLayer("Counties");

  // If function was called with 'Off', make the
  // following layers invisible...
  if (type == 'Off')
  {
    for (i=2; i<20; i++;)
    {
      MyLayer[i].setVisibility(false);
    }
  }
  // ...otherwise, make the following layers visible
  else
  {
    for (i=2; i<20; i++;)
    {
      MyLayer[i].setVisibility(true);
    }
  }
  // Tell the Viewer to rebuild the following layers
  // when the map is refreshed
  for (i=2; i<20; i++;)
  {
    MyLayer[i].setRebuild(true);
  }
  // Refresh the map
```

Municipal Application (*continued*)

```
    myMap.refresh();

  // End the if statement that verified not busy
  }
  // If Viewer is busy, don't do the stuff above;
  // instead, display alert
  else
     alert("The Viewer is busy. Please try again in a few seconds.");

// End the function
}

// =-=========-=-=========-=-=========-=-=========-=-=========-=-= //
// Function: raster(type)
// Description: Turns raster layer on or off. (called by
// ON/OFF buttons, under Layer Control: Raster Layers)
// Arguments: string ('On' or 'Off')
// Returns: nothing
// =-=========-=-=========-=-=========-=-=========-=-=========-=-= //
function raster(type)
{
  myMap = getMyMap();            // Get instance of MGMap

  // If Viewer is not busy get current map scale, then assign
  // "San Rafael" layer to MyRastLayer variable...
  if (myMap.isBusy() == false)   {
    var CurrentScale = myMap.getScale();
    var MyRastLayer = myMap.getMapLayer("San Rafael");

    // If scale is less than 1:20,000 and if function was called
    // with 'On', make MyRastLayer visible
    if (CurrentScale < 20000 && type == 'On')
    {
      MyRastLayer.setVisibility(true);
      MyRastLayer.setRebuild(true);
      myMap.refresh();
    }
    // If scale is less than 1:20,000 and if function was called
    // with 'Off', make MyRastLayer invisible.
    if (CurrentScale < 20000 && type == 'Off')
    {
      MyRastLayer.setVisibility(false);
      MyRastLayer.setRebuild(false);
      myMap.refresh();
    }
  }
  // If Viewer is busy, don't do the stuff above;
  // instead, display alert
  else
```

Municipal Application (*continued*)

```
      alert("The Viewer is busy. Please try again in a few seconds.");
}

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: digit()
// Description: Lets users create report data for a specified
// point. (called by the 'Digitize Incident' button, under
// 'Incident Entry & Reporting')
//
// NOTE: This function just gathers the point coordinates,
// fires the onDigitizedPoint event, and passes the coordinates
// to that event. The event is linked to a function (defined
// in a separate frame -- see www.autodesk.com/mapguidedemo for
// the source) that runs a ColdFusion file and creates a new
// window to hold the ColdFusion-generated HTML output. The HTML
// output includes a form that lets enter text and add that text
// to the map as point data. See Chapter 4 for more information
// about ColdFusion.
//
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function digit()
{
  // Use browserId variable (defined at beginning of script)
  // to determine if user has Netscape or Internet Explorer;
  // doesn't bother to call getMyMap(), because entire function
  // varies by browser

  // If Netscape...
  if (browserId == 1)
  {
    // Get instance of MGMap, assign to MyMap variable
    MyMap = parent.main.document.embeds[0];

    // If Viewer is not busy, call digitizePoint() method;
    // otherwise display alert (because digitizePoint() fires
    // the onDigitizedPoint event, we must pass the observer
    // as a function argument)

    if (myMap.isBusy() == false)
      MyMap.digitizePoint(parent.rightempty.document.obs);
    else
      alert("The Viewer is busy. Please try again in a few sec-
onds.");
  }
  // If Internet Explorer...
  if (browserId == 2)
  {
```

For more info...
See *Chapter 4, "Using Reports to Query and Update Data Sources."*

Municipal Application (*continued*)

```
    // Get instance of MGMap, assign to MyMap variable
    MyMap = parent.main.document.MyMap;

    // If Viewer is not busy, call digitizePoint() API method;
    // otherwise display alert
    if (myMap.isBusy() == false)
      MyMap.digitizePoint();
    else
      alert("The Viewer is busy. Please try again in a few sec-
onds.");
  }
}

// =-=========-=-=========-=-=========-=-=========-=-=========-= //
// Function: reporter()
// Description: Generates report data for selected map features
// (called by 'Reporter' button, under 'Incident Entry
// & Reporting)
// Arguments: none
// Returns: nothing
// =-=========-=-=========-=-=========-=-=========-=-=========-= //
function reporter()
{
  myMap = getMyMap();            // Get instance of MGMap

  // If Viewer is not busy...
  if (myMap.isBusy() == false)
  {
    // Get object representing current selection, assign; then
    // get number of map features in that selection
    var MySel = MyMap.getSelection();
    var NumSel = MySel.getNumObjects();

    // If selection has at least one object, display View Reports
    // dialog; otherwise display alert
    if (NumSel > 0)
      MyMap.viewReportsDlg();
    else
      alert("You need to select map features before you can
          generate a report.");
  }
  // If Viewer is busy, don't do the stuff above;
  // instead, display alert
  else
    alert("The Viewer is busy. Please try again in a few seconds.");
}

// =-=========-=-=========-=-=========-=-=========-=-=========-= //
```

Municipal Application (*continued*)

```
// Function: showIncidents()
// Description: Constructs SQL 'WHERE' statement requesting map
// features (request is based on user's selection from 'Date' and
// 'Status' drop-downs, under 'Incident Entry & Reporting'); sends
// the SQL statement to the database that's linked to the 'Incidents'
// layer; then refreshes the map, causing the requested features
// to display
// Arguments: none
// Returns: none
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function showIncidents()
{
  myMap = getMyMap();            // Get instance of MGMap

  // If Viewer is not busy...
  if (MyMap.isBusy() == false)
  {
    // If user has old copy of Viewer
    // display alert only
    var ApiVersion = MyMap.getApiVersion();
    if (ApiVersion < 5.0)
    {
      alert("This control uses the latest technology in the\n
           Autodesk MapGuide Viewer Release 5 API. Please \n
           download the latest Viewer from the Autodesk MapGuide\n
           web site (www.autodesk.com/mapguide).");
    }
    // If user has recent copy of Viewer (API version 5.0
    // or greater)...
    else
    {
      // Assign array of possible drop-down list options
      // to selValue variable; then assign name of selected
      // list item to 'temp' variable ('Selection' is the
      // HTML form, 'Status' is the 'status:' drop-down list)
      var selValue = document.Selection.Status.options;
      for (var i = 0; i < selValue.length; i++)
      {
        if (selValue[i].selected)
        {
          var temp = selValue[i].value;
        }
      }
      // If user selected 'Show All', assign text string to
      // whereClause1 varible
      if (temp == 'showAll')
      {
        var whereClause1 = "Status is not null";
      }
```

Municipal Application (*continued*)

```
// If user selected any other drop-down item, assign
// whereClause1 the string "Status=" plus the list item name
// (i.e., "Status='New'" or "Status='Old'")
else
{
  var whereClause1 = "Status='" + temp + "'";
}

// Assign array of possible drop-down list options
// to selValue variable; then assign name of selected
// list item to 'temp2' variable ('Selection' is the
// HTML form, 'myDate' is the 'date:' drop-down list)
var selValue = document.Selection.myDate.options;
for (var i=0; i < selValue.length; i++)
{
  if (selValue[i].selected)
  {
    var temp2 = selValue[i].value;
  }
}
// The temp2 variable now represents user's selection -- use
// it to assign appropriate date-related SQL statement to the
// whereClause2 variable (date is generated on the server
// by Coldfusion, then sent to the browser as text)
if (temp2 == 'today')
{
  whereClause2 =
    "ReportDate <= #04/01/99# AND ReportDate >= #04/01/99#"
}
else if (temp2  == 'last2days')
{
  whereClause2 =
    "ReportDate <= #04/01/99# AND ReportDate >= #03/30/99#"
}
else if (temp2  == 'last7days')
{
  whereClause2 =
    "ReportDate <= #04/01/99# AND ReportDate >= #03/25/99#"
}
else if (temp2  == 'last15days')
{
  whereClause2 =
    "ReportDate <= #04/01/99# AND ReportDate >= #03/17/99#"
}
else if (temp2  == 'last30days')
{
  whereClause2 =
    "ReportDate <= #04/01/99# AND ReportDate >= #03/02/99#"
}
```

Municipal Application (*continued*)

```
        else
        {
          whereClause2 = "ReportDate is not null";
        }

        // Combine the two SQL statements into one
        whereClause = ((whereClause1) + " AND " + (whereClause2));

        // Create an object representing the database setup for
        // the 'Incidents' layer, then assign that object to
        // a variable called 'MyDatabaseSetup'
        var MyMapLayer = MyMap.getMapLayer("Incidents");
        var MyLayerSetup = MyMapLayer.getLayerSetup();
        var MyDatabaseSetup = MyLayerSetup.getDatabaseSetup();

        // Run SQL statement you created on the database linked
        // to the 'Incidents' layer; then refresh the map, causing
        // the items you queried to display in the map.
        MyDatabaseSetup.setWhereClause(whereClause);
        MyMap.refresh();
      }
    }
}
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: pageSetup()
// Description: Displays the page setup dialog (called by
// the 'Print Setup' button, under 'Print Map')
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function pageSetup()
{
  myMap = getMyMap();            // Get instance of MGMap

  // If Viewer is not busy, display Page Setup dialog;
  // otherwise display alert
  if (myMap.isBusy() == false)
    myMap.pageSetupDlg();
  else
     alert("The Viewer is busy. Please try again in a few seconds.");
}

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function printMap()
// Description: Displays Print dialog (called by the
// 'Print' button, under 'Print Map')
// Arguments: none
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
```

Municipal Application (*continued*)

```
function printMap()
{
  myMap = getMyMap();            // Get instance of MGMap

  // If Viewer is not busy, display Print dialog;
  // otherwise display alert
  if (myMap.isBusy() == false)
    myMap.printDlg();
  else
     alert("The Viewer is busy. Please try again in a few seconds.");
}

//-->
</SCRIPT>

<-! Rest of page is straight HTML, with the exception of FORM
elements that call the JavaScript functions defined above. ->
</HEAD>
<BODY BGCOLOR="#CCCC99">
<-! Remainder of page is FORM containing a nested table ->
<FORM NAME="Selection">

<-! Begin table 1 ->
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0 WIDTH=100%>
<TR>
<TD VALIGN=top width=100%>

  <-! Begin table 2 ->
  <TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
  <TR>
  <TD BGCOLOR="#FFFFFF" VALIGN=MIDDLE ALIGN=CENTER>
  <IMG SRC="menu/mdcp.gif" WIDTH=105 HEIGHT=31 BORDER=0><BR>
  </TD>
  </TR>
  </TABLE>
  <-! End table 2 ->

  <-! Begin table 3 ->
  <TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
  <TR>
  <TD VALIGN=MIDDLE ALIGN=left BGCOLOR="#9c9c63">
  <A HREF="app98help.cfm/#incident"><IMG SRC="MENU/qt.gif"
    WIDTH=21 HEIGHT=29 BORDER=0 align=right></A>
  <IMG SRC="MENU/incmon.gif" WIDTH=56 HEIGHT=29 BORDER=0><BR>
  </TD>
  </TR>
  <TR>
  <TD VALIGN=MIDDLE ALIGN=CENTER>
```

Municipal Application (*continued*)

```
    <-! Begin table 4 ->
    <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
    <TR>
    <TD>
    <IMG SRC="menu/noti.gif" WIDTH=68 HEIGHT=21><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:notify()"><IMG SRC="MENU/ON.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:notifyoff()"><IMG SRC="MENU/OFF.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    </TR>
    </TABLE>
    <-! End table 4 ->

</TD>
</TR>
</TABLE>
<-! End table 3 ->

<-! Begin table 5->
<TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
<TR>
<TD VALIGN=MIDDLE BGCOLOR="#9c9c63">
<A HREF="app98help.cfm/#digimg"><IMG SRC="MENU/qs.gif"
  WIDTH=21 HEIGHT=18 BORDER=0 ALIGN=right></A>
<IMG SRC="menu/lc.gif" WIDTH=93 HEIGHT=18 BORDER=0><BR>
</TD>
</TR>
<TR>
<TD VALIGN=MIDDLE ALIGN=CENTER>

  <-! Begin table 6 ->
  <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
  <TR>
  <TD>
  <IMG SRC="menu/incident.gif" WIDTH=75 HEIGHT=21><BR>
  </TD>
  <TD>
  <A HREF="JavaScript:incidents('On')"><IMG SRC="MENU/ON.gif"
    WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
  </TD>
  <TD>
  <A HREF="JavaScript:incidents('Off')"><IMG SRC="MENU/OFF.gif"
    WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
  </TD>
```

Municipal Application (*continued*)

```
    </TR>
    </TABLE>
    <-! End table 6 ->

    <-! Begin table 7 ->
    <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
    <TR>
    <TD>
    <IMG SRC="menu/vector.gif" WIDTH=75 HEIGHT=21><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:vector('On')"><IMG SRC="MENU/ON.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:vector('Off')"><IMG SRC="MENU/OFF.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    </TR>
    </TABLE>
    <-! End table 7 ->

    <-! Begin table 8 ->
    <TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
    <TR>
    <TD>
    <IMG SRC="menu/raster.gif" WIDTH=75 HEIGHT=21><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:raster('On')"><IMG SRC="MENU/ON.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    <TD>
    <A HREF="JavaScript:raster('Off')"><IMG SRC="MENU/OFF.gif"
      WIDTH=30 HEIGHT=21 BORDER=0></A><BR>
    </TD>
    </TR>
    </TABLE>
    <-! End table 8 ->

</TD>
</TR>
</TABLE>
<-! End table 5 ->

<-! Begin table 9 ->
<TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
<TR>
<TD BGCOLOR="#9c9c63">
```

Municipal Application (*continued*)

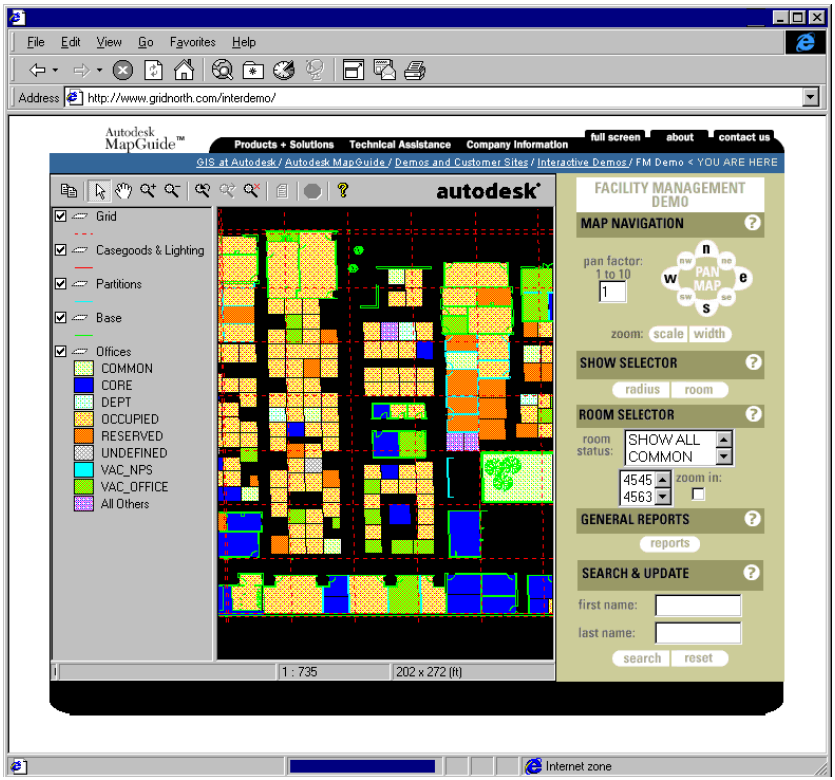
```
<A HREF="app98help.cfm/#ier"><IMG SRC="MENU/qt.gif"
  WIDTH=21 HEIGHT=29 BORDER=0 align=right></A>
<IMG SRC="menu/incen.gif" WIDTH=95 HEIGHT=29 BORDER=0>
</TD>
</TR>
<TR>
<TD VALIGN=MIDDLE ALIGN=CENTER>
<a HREF="JavaScript:digit();"><IMG SRC="menu/diginc.gif"
  WIDTH=107 HEIGHT=19 BORDER=0></a>
<a href="JavaScript:reporter()"><IMG SRC="MENU/reporter.gif"
  WIDTH=107 HEIGHT=19 BORDER=0></a>
</TD>
</TR>
</TABLE>
<-! End table 9 ->

<-! Begin table 10 ->
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
<TD COLSPAN="1" ALIGN="CENTER"><IMG SRC="menu/b_date.gif"
  WIDTH=27 HEIGHT=12 ALT="" BORDER="0"></TD>
<TD>
<SELECT NAME="myDate" SIZE="1" onChange="showIncidents();">
  <OPTION VALUE="showAll">Show all
  <OPTION VALUE="today">Today
  <OPTION VALUE="last2days">Last 2 days
  <OPTION VALUE="last7days">Last 7 days
  <OPTION VALUE="last15days">Last 15 days
  <OPTION VALUE="last30days">Last 30 days
</SELECT>
</TD>
</TR>
<TR>
<TD>
<IMG SRC="menu/b_status.gif" WIDTH=36 HEIGHT=10 ALT=""
  BORDER="0"></TD>
<TD COLSPAN="2">
<SELECT NAME="Status" SIZE="1" onChange="showIncidents();">
  <OPTION VALUE="showAll">Show all
  <OPTION VALUE="New">New
  <OPTION VALUE="Open">Open
</SELECT>
</TD>
</TR>
</TABLE>
<-! End table 10 ->

<-! Begin table 11 ->
<TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
```

Municipal Application (*continued*)

```
  <TR>
  <TD BGCOLOR="#9c9c63">
  <A HREF="app98help.cfm/#mapprn"><IMG SRC="MENU/qs.gif"
    WIDTH=21 HEIGHT=18 BORDER=0 align=right></A>
  <IMG SRC="menu/prnmap.gif" WIDTH=64 HEIGHT=18 BORDER=0><BR>
  </TD>
  </TR>
  <TR>
  <TD VALIGN=MIDDLE ALIGN=CENTER>
  <A HREF="JavaScript:pageSetup()"><IMG SRC="MENU/prnsetup.gif"
    WIDTH=107 HEIGHT=19 BORDER=0></a><BR>
  <A HREF="JavaScript:printMap()"><IMG SRC="MENU/print.gif"
    WIDTH=107 HEIGHT=19 BORDER=0></a><BR>
  </TD>
  </TR>
  </TABLE>
<-! End table 11 ->

</TD>
</TR>
</TABLE>
<-! End table 1->

</FORM>
</BODY>
</HTML>
```

Facility Management Application

The Facility Management (FM) application demonstrates how you can create a web-based facility management application to manage and maintain various facilities. Its layout is similar to the municipal application, but it has more advanced navigation controls on the right. It also allows you to select features in various ways, generate reports, and even search for an employee and update his or her information.



Facility Management application

Source Code

Following is the source code for the controls. Additional comments have been added to the code to give you a better idea of how the scripting works. To view the source code for the other frames in this application, go to the Demos and Customers section of the Autodesk MapGuide web site (*www.autodesk.com/mapguidedemo*), click Interactive Demos, and then click the Facility Management application. When it is loaded, you can use your browser's View Source command to view the complete code behind the page.

Facility Management Application

```
<HTML>
<HEAD>
<TITLE>FM</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
<!--
// Get full browser name and assign it to tempName variable;
// then assign first 8 letters of tempName to browserName variable
var tempName = navigator.appName;
var browserName = tempName.substring(0,8);

// Map object variable, to be used later
var MyMap;

// Set browserId variable: '1' for netscape, '2' for IE, else '0'
if (browserName == 'Netscape')
  var browserId = 1;
else if (browserName == 'Microsof') // just first 8 letters...
  browserId=2;
else
  browserId = 0;

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: getMyMap()
// Description: Get appropriate map object for IE or Netscape
// **Same concept as getThisMap() example on page 15)**
// Arguments: none
// Returns: map object
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function getMyMap()
{
  // Get appropriate MGMap object; type depends on browserId value
  if (browserId == 1)
    MyMap = top.main.document.embeds[0]; // Netscape map object
  else if (browserId == 2)
    MyMap = top.main.document.MyMap;     // IE map object
  else
    MyMap = null;                        // none if other browser
```

Facility Management Application (*continued*)

```
  return MyMap;                              //return map object
}


// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: Pan(direction)
// Description:pans in the specified direction
// Arguments:direction
// Return:nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function Pan(direction)
{
   // Get MGMap object
   var MyMap = getMyMap();
  MyMap.setAutoRefresh(false);
   // Get Width or Height in meters
   var delta;
  var scrollfactor = 1;
   scrollfactor = document.Selection.factor.value;
   scrollfactor = parseInt(scrollfactor);
   if (isNaN (scrollfactor) || (scrollfactor < 1)) {
     alert("Enter a positive number as the scrolling factor.");
  }
   else
  {
     if (direction == 'Up' | direction == 'Down' | direction ==
     'Left' | direction == 'Right' | direction == 'Ul' | direction
     == 'Ur' | direction == 'Ld' | direction == 'Rd')
      delta = MyMap.getWidth("M");
      delta = (scrollfactor/10) * delta;
   // Compute center point of map in Mapping Coordinate System (MCS)
      var xyPt = MyMap.lonLatToMcs(MyMap.getLon(), MyMap.getLat());
      // Convert delta from Meters to MCS units.
      var MCStoMeters = MyMap.getMCSScaleFactor();
      delta = delta / MCStoMeters;
      // Adjust by width / height of the map
      if (direction == 'Left') {
         xyPt.setX(xyPt.getX() - delta)
      }
      if (direction == 'Right') {
         xyPt.setX(xyPt.getX() + delta)
      }
      if (direction == 'Up') {
         xyPt.setY(xyPt.getY() + delta)
      }
      if (direction == 'Ul') {
         xyPt.setX(xyPt.getX() - delta)
         xyPt.setY(xyPt.getY() + delta)
      }
```

Facility Management Application (*continued*)

```
     if (direction == 'Ur') {
        xyPt.setX(xyPt.getX() + delta)
        xyPt.setY(xyPt.getY() + delta)
     }
     if (direction == 'Down') {
        xyPt.setY(xyPt.getY() - delta)
     }
     if (direction == 'Ld') {
        xyPt.setX(xyPt.getX() - delta)
        xyPt.setY(xyPt.getY() - delta)
     }
     if (direction == 'Rd') {
        xyPt.setX(xyPt.getX() + delta)
        xyPt.setY(xyPt.getY() - delta)
     }
     // Zoom to the new location
     myScale = MyMap.getScale();
     MyMap.zoomScale(xyPt.getY(), xyPt.getX(), myScale);
     MyMap.setAutoRefresh(true);
     MyMap.refresh();
   } //ends panning
}
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: GoToOrig(lat, lon)
// Description:Zooms to the lat lon specified with a width of 400
ft// Arguments:lat, lon
// Returns: nothing
// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
function GoToOrig(lat, lon){
  var MyMap = getMyMap();
  MyMap.zoomWidth(lat, lon, 400, "FT");
}

// =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-= //
// Function: reportsDlg()     +
// Description:shows the reports dialog for generating reports+
// Arguments:none                  +
// Return:none                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
function reportsDlg(){
    var MyMap = getMyMap();
   var MyMapSel = MyMap.getSelection();
   var MyMapLayer = MyMap.getMapLayer("Offices");
  if ((MyMapSel.getNumObjects() < 1) || (MyMapLayer.isVisible() ==
false)){
     alert("You must select an office first.");
   }
   else{
```

Facility Management Application (*continued*)

```
    MyMap.viewReportsDlg();
  }
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+  Function: zoom(type)          +
+  Description:general use of zoom functions+
+  Arguments:type string         +
+  Return:null                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
function zoom(type)
{
  var myMap = getMyMap();//get MGMap
  if (myMap.isBusy() == false)
  {
    if (type == 'Scale')
      myMap.zoomScaleDlg();
    else
      myMap.zoomWidthDlg();
  }
  else
    alert("The Viewer is busy ... please\n try again in a few sec-
onds");
}


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+  Function: selMapObj()         +
+  Description:select by object, uses the Select Map Objects dialog
+
+  Arguments:none                   +
+  Return:none                         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
function selMapObj(){
    var MyMap = getMyMap();
  MyMap.selectMapObjectsDlg();
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+  Function: selRadiusMode()  +
+  Description:changes the selection mode to select by radius+
+  Arguments:none                   +
+  Return:none                         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
```

Facility Management Application (*continued*)

```
function selRadiusMode(){
    var MyMap = getMyMap();
  MyMap.selectRadiusMode();
}

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+  Function: ObjSelChanged()  +
+  Description:on change of selection, highlight room selected+
+  Arguments:none                  +
+  Return:none                       +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
function ObjSelChanged()
{
  // Get MGMap object
   var MyMap = getMyMap();
    if (MyMap.isBusy() == false){
     var selOptions = document.Selection.roomnum.options;
    var MyCollection = MyMap.createObject("MGCollection");
    var MyMapSel = MyMap.getSelection();
    var MyMapLayer = MyMap.getMapLayer("Offices");
    // For each item selected in the list box, get the corresponding
    // object from the map.  Keep track of them in a vector
    for (var i=0; i < selOptions.length; i++) {
      if (selOptions[i].selected){
          var MyObj = MyMapLayer.getMapObject(selOptions[i].val-
ue);
            if (MyObj != null) {
              MyCollection.add(MyObj);
               }
         }
     }
     MyMapSel.clear();
     if (MyCollection.size() > 0){
      MyMapSel.addObjectsEx(MyCollection, false);
     }
    var zoomCheck = document.Selection.ZoomOption.checked;
    if (zoomCheck == true){
      MyMap.zoomSelected();
    }
  }
}


/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+  Function: showOccupancy()  +
+  Description:on change of selection, show occupancy type+
```

Facility Management Application (*continued*)

```
+  Arguments:none                  +
+  Return:none                       +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++*/
function showOccupancy()
{
  // Get MGMap object
   var MyMap = getMyMap();
    if (MyMap.isBusy() == false){
     var selValue = document.Selection.Status.options;
      for (var i=0; i < selValue.length; i++) {
      if (selValue[i].selected){
          var temp = selValue[i].value;
         }
     }
    if (temp == 'showAll'){
      var whereClause = "Space_Status is not null";
    }
    else{
      var whereClause = "Space_Status='"+temp+"'";
    }
     var MyMapLayer = MyMap.getMapLayer("Offices");
    MyMapLayer.setSQLWhere(whereClause);
    MyMap.refresh();
  }
}


/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+ FunctionresetForm()              +
+ Description:resets the values in the form to the default values+
+ Arguments:none                   +
+ Return:none                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++*/
function resetForm(){
  document.Selection.reset();
}

/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
+ FunctionopenSearchWind()    +
+ Description:used to launch the search popup window+
+ Arguments:none                   +
+ Return:none                        +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++*/
function openSearchWind(){
```

Facility Management Application (*continued*)

```
  var FirstName = document.Selection.FirstName.value;
  var LastName = document.Selection.LastName.value;
  var RoomSelected;
  var Count = 0;
   var selOptions = document.Selection.roomnum.options;
    for (var i=0; i < selOptions.length; i++) {
    if (selOptions[i].selected){
       RoomSelected = selOptions[i].value
       Count = Count + 1;
       }
    }
  if ((FirstName.length == 0) && (LastName.length == 0) && (Count <
1)){
     alert("Must enter a value for the first name or last name field.
\nOr, select a room before continuing.");
  }
  else{
     SearchWindow = window.open("Search.cfm?FirstName="+First-
Name+"&LastName="+LastName+"&Room="+RoomSelected, "SearchWindow",
"toolbar=no,width=350,height=205,directories=no,status=no,scroll-
bars=no,resize=yes,menubar=no")
  }
}

//-->
</SCRIPT>
</HEAD>
```

Using Reports to Query and Update Data Sources

In This Chapter

4

- Autodesk MapGuide reports
- Introducing ColdFusion and ASP
- Creating report scripts with ColdFusion
- Creating report scripts with ASP

Your Autodesk MapGuide applications can include reports that enable users to display and modify database information associated with a map. This chapter explains how Autodesk MapGuide generates reports and shows you how to create report scripts using two popular server-side technologies, Allaire ColdFusion and Microsoft Active Server Pages.

Autodesk MapGuide Reports

When creating a map, you can add *reports* to the map. Typically, a report is an HTML page that displays information about the selected map features on the layer. However, because the power behind the report is a report script that you create using a third-party tool like ColdFusion, Active Server Pages (ASP), Java, LiveWire™, or dbWeb™, the report can do much more than display information—it can perform any number of tasks that you code into the script. For example, in this chapter you will see a sample application that allows the user to click a point on the map and then updates the source database with that point, so that any map layer that uses that database as its data source will now display that point on the map. This chapter focuses on these types of advanced applications performed by the report script.

How Reports are Generated

Autodesk MapGuide's role in generating reports is to construct a URL dynamically and send it as an HTTP request to a web server. This URL is composed of a path to an application on the web server along with a set of parameters. The server, in turn, will process the request and send or "post" the results.

Autodesk MapGuide can generate two distinct types of requests by passing unique parameters along with the URL to the server. The first type of URL request passes key values of the selected map features. These key values are the "keys" that are defined in the data source. The second type of URL request passes a point feature and its location.

Specifying the Report Script

The report script contains the necessary code to connect to the appropriate database, build the query, and display the results. For example, the script might be a ColdFusion template file (CFM) or Active Server Page (ASP) that resides on the web server. The Reports tab in the Map Window Properties dialog box in Autodesk MapGuide Author allows you to specify the report script, as well as set other properties of the report. In the URL text box, you specify the name and path of the script to use to pass the report information to your reporting engine.

The Request

Autodesk MapGuide Author will use the report settings defined in the Map Window Properties dialog box to construct the URL that is sent to the server. For example, after all of the settings are specified, the URL might look like this:

```
http://www.myserver.com/reports/
  report.cfm&OBJ_TYPE=landuse&OBJ_KEYS='01235639','01235640','01235641'
```

This URL is a request to launch a ColdFusion template called *report.cfm*. The template file and the two parameters OBJ_TYPE and OBJ_KEYS are passed to the ColdFusion engine by the web server. The parameters will serve as arguments or variables that can be used by the ColdFusion template file.

Note By default, Autodesk MapGuide Author sends map feature key values to the URL as characters. However, if you specify another data type for the key column, Autodesk MapGuide will send the keys as that type instead. You set the key column type by selecting it from the Type list box on the Data Sources tab of the Map Layer Properties dialog box.

Launching the Report

For more info...
See Chapter 2 and the *Autodesk MapGuide Viewer API Help* for more information on the Viewer API.

There are several ways to launch the View Reports dialog box from Autodesk MapGuide Viewer. You can right-click the map and choose View ➤ Reports, or you can click the Report button on the Autodesk MapGuide Viewer toolbar. Both methods will display a dialog box that shows a list of available reports defined for the map. Using the Autodesk MapGuide Viewer API, you can also launch reports programmatically; you call the View Reports dialog box using viewReportsDlg and launch the report directly using viewReport.

Introducing ColdFusion and ASP

The examples in this chapter were created using two report engines, Allaire ColdFusion and Microsoft Active Server Pages (ASP). ColdFusion and ASP are *application servers*. An application server is an application that works with the web server to provide additional web functionality. Like the web server, it runs in the background as a Windows NT service.

Both products work essentially the same way. You build web pages that include special tags, and when a web browser requests one of those pages, the application server interprets the tags, replaces them with the results of the specified calculations or database queries, and then sends the completed page to the web server. The web server then sends the page to the browser to

be displayed. Because the processing is done by the server, the end-user sees only the final HTML output, not the code used to create that output. (Of course, the HTML can include anything—even client-side scripting code!) Although end-users can view the source of your HTML output, they never see the server-side scripting code used to create that output.

For more info...
Refer to the *Autodesk MapGuide User's Guide* for information about installing a report engine.

This book uses ColdFusion and ASP for its examples because developing with these products is easier than writing your own perl scripts or Visual Basic/C++ DLLs, and because these products are by far the most common platforms for Autodesk MapGuide server-side application development. Both are free to Autodesk MapGuide developers: ColdFusion is included on the Autodesk MapGuide Server CD, and ASP is offered by Microsoft at no cost as part of Windows NT Server Option Pack 4. Although the examples are specific to ColdFusion and ASP, the concepts are general, applying to CGI and to other application servers as well.

ColdFusion supports both Microsoft Internet Information Server (IIS) and the Netscape web servers. ASP supports IIS only, meaning that it, and your map applications, can only be run on the Microsoft web server. Keep in mind, though, that this does not affect your users; the HTML you produce can be read by any web *browser*. The limitation exists only for the web *server*.

Creating Report Scripts with ColdFusion

For more info...
Refer to the documentation that shipped with your copy of ColdFusion, or to the online documentation at: *www.allaire.com*

A ColdFusion script, or *template*, is essentially a standard HTML file that includes extra tags written in a server-side markup language called CFML (ColdFusion Markup Language). CFML tags begin with the letters CF, and are used to tell ColdFusion to process either a calculation or a query. The tags can also tell ColdFusion which data source you want to use and how you want to manipulate or display the information in that data source. A template uses the file extension *.cfm* to identify itself and let the web server know that it should be passed to the ColdFusion service for processing.

ColdFusion was designed to provide database connectivity to your web pages. It is a full-fledged development environment that includes functions, operators, variables, control structures, and more. You can use ColdFusion to create powerful and complex web applications. But simple applications have their uses too, as we'll see later in this chapter. Despite its power, ColdFusion is fairly easy to learn; if you're familiar with HTML coding, you'll get up to speed quickly.

The following examples show how to create report scripts with ColdFusion. We recommend that you read them in order.

Note For ASP versions of the same examples, see "Creating Report Scripts with ASP" on page 102.

Example—Listing File Contents with ColdFusion

This example shows a simple template that lists the contents of a map-resource database. Note that this template accesses the database directly, instead of using Autodesk MapGuide's reporting feature. Later, we'll see how Autodesk MapGuide fits into the picture.

Let's say you have an MWF file that points to a database containing parcel information such as lot number, street address, owner's name, and so on. If you want to list the contents of that database at the bottom of an HTML page displaying the map, you would first rename the HTML file with a *.cfm* extension and place it in a directory with script or execute permissions. Then you would add <CFQUERY> and <CFOUTPUT> statements to the file. The <CFQUERY> tag tells ColdFusion which database to use and which records to select from that database. You can place <CFQUERY> anywhere in the page, as long as it appears before <CFOUTPUT>. The <CFOUTPUT> tag controls how the database output will be displayed on the page. You place this tag within <BODY>, at the location you want the database output to appear.

The next sections describe each of these tasks in more detail.

Setting up the Query

First we'll build the <CFQUERY> statement. If your map links to a table called Parcel_Data through a data source Assessor, <CFQUERY> will look like this:

```
<CFQUERY NAME="get_parcel_info" DATASOURCE="Assessor">
   SELECT * FROM Parcel_Data
</CFQUERY>
```

For more info...
Refer to the Autodesk MapGuide online help for more information on OLE DB.

The NAME attribute specifies the name of the ColdFusion query. This name can be anything you want, as long as it matches the name specified later in <CFOUTPUT>. The DATASOURCE attribute is the OLE DB data source name (DSN), in this case "Assessor". Between the <CFQUERY> beginning and end tags is a SQL statement specifying which part of the table you want to look at (this selection is known as a *recordset*.) In this case, we're selecting everything ("*") from the Parcel_Data table.

Controlling the Output

Now we'll assemble the <CFOUTPUT> statement. If you want to display the parcel number, owner's name, and year built, your tag will look like this:

```
<CFOUTPUT QUERY="get_parcel_info">
   <P>Parcel Number: #APN#<BR>
   <P>Owner Name: #Owner_Name#<BR>
   <P>Year Built: #Year_Built#</P>
</CFOUTPUT>
```

The QUERY attribute tells ColdFusion which recordset you'd like to display; this attribute matches the NAME you specified in <CFQUERY>. The names within pound signs (#APN#, #Owner_Name#, #Year_Built#) are ColdFusion variables that match column names in the database table (for example, #APN# refers to the APN column). Everything else is straight HTML.

Seeing the Results

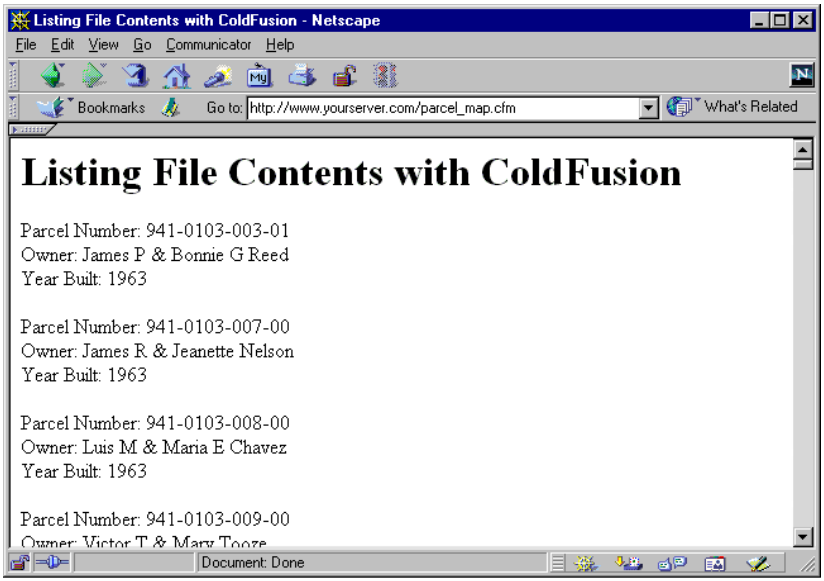
Now we're ready to load the page in the browser.

However, because this particular table has more than 5,000 records, selecting everything in it might not be such a good idea. Let's limit the output by showing only houses built in 1963. To do so, go back to <CFQUERY> and change the SQL statement to the following:

```
SELECT * FROM Parcel_Data WHERE Year_Built = '1963'
```

That's still a lot of records, but probably not enough to generate angry e-mail messages from users. Here's a listing of the complete CFM file called *parcel_report.cfm*, followed by an illustration of the page as it appears in a browser:

```
<HTML>
<HEAD>
<!-- ColdFusion query -->
<CFQUERY NAME="get_parcel_info" DATASOURCE="Assessor">
SELECT * FROM Parcel_Data WHERE Year_Built = '1963'
</CFQUERY>
<TITLE>ColdFusion Example</TITLE>
</HEAD>
<BODY>
<H1>Listing File Contents with ColdFusion</H1>
<!-- ColdFusion output tags -->
<CFOUTPUT QUERY="get_parcel_info">
<P>Parcel Number: #APN#<BR>
Owner: #Owner_Name#<BR>
Year Built: #Year_Built#</P>
</CFOUTPUT>
</BODY>
</HTML>
```



The HTML output

In this example, the database happens to be an Autodesk MapGuide resource, but it could really be anything: an Access database listing employees and their phone numbers, an Excel spreadsheet showing your checking account balance, or anything else you might store in a table. In most cases, you'll want to access your database resources through Autodesk MapGuide Viewer, by linking them to features and layers in the map. The next two examples show you how to do this.

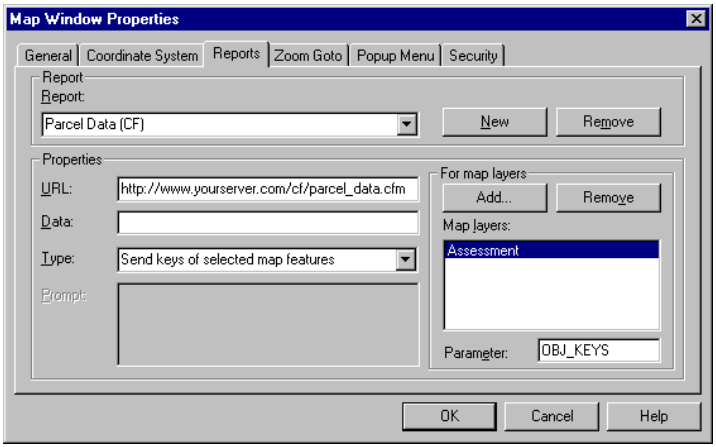
Example—Querying and Displaying Data via the Map

Now that we've seen how ColdFusion works, let's use it with Autodesk MapGuide. This example uses *StarterApp.mwf* from the Autodesk MapGuide web site. Note that the full set of Starter Application files is available for download at *http://www.autodesk.com/mapguidedemo*.

Setting Up the Report in Autodesk MapGuide Author

We'll start by specifying report information in Autodesk MapGuide Author. Our report will be called "Parcel Data (CF)", and it will access a CFM file whose URL is *http://www.yourserver.com/parcel_report.cfm*. We want the CFM file to display information about user-selected features on a layer called

"Assessment." The Map Window Properties dialog box shown next reflects our selection.



Dialog box specifications for 'Parcel Data (CF)'

Here are descriptions of how we used the options on the Reports tab:

Report	Specifies the report name as it appears in the Autodesk MapGuide Viewer. Our report is "Parcel Data (CF)."
URL	Specifies the name and location of the report script, in this case *parcel_report.cfm* on *www.yourserver.com*.
Data	We left this field blank but could have used it to pass additional URL parameters to *parcel_report.cfm*. For example, if our ColdFusion file contained definitions for more than one query, we might have passed a parameter telling the file which of the queries to run, such as report = 'A'.
Type	Specifies whether the report is based on the keys of selected features (as this one is), or on the coordinates of a point the user clicks.
For Map Layers	Specifies the layer or layers you want the report to be linked to. Our report operates only on features on the Assessment layer.
Parameter	Specifies the name of the URL parameter used to send the feature key (or keys) to *parcel_report.cfm*. The name can be anything you want, as long as it matches the name you specified in *parcel_report.cfm*. We've selected the Autodesk MapGuide Author default, "OBJ_KEYS".

When a user selects one or more features from the Assessment layer and runs the Parcel Data (CF) report, Autodesk MapGuide constructs a URL that invokes *parcel_report.cfm* and tells it to generate a report on the selected features, which are identified by their OBJ_KEY values. If the user selected a single feature whose key was "941-0176-003-00", the URL would look like this:

```
http://www.yourserver.com/parcel_report.cfm?OBJ_KEYS='941-0176-003-00'
```

If the user selected multiple features, the URL might look like this:

```
http://www.yourserver.com/
  parcel_report.cfm?OBJ_KEYS='941-0176-003-00','941-0176-006-00','941-0176-004-00'
```

Note that OBJ_KEYS is represented as a standard URL parameter. To ColdFusion, this parameter is no different from one submitted by an HTML form element. As we'll see in the next section, ColdFusion processes it accordingly.

Creating the Report Script

Now let's create the ColdFusion template that will process the Autodesk MapGuide report. The following listing is for the *parcel_report.cfm* file:

```
<HTML>
<HEAD><TITLE>ColdFusion Report Data</TITLE></HEAD>
<BODY>
ColdFusion-- ColdFusion query -->
<CFQUERY DATASOURCE="assessor" NAME="get_parcel_info">
   SELECT * FROM Parcel_Data WHERE APN IN (#PreserveSingleQuotes(OBJ_KEYS)#)
</CFQUERY>
<H1>ColdFusion Report Data</H1>
<!-- ColdFusion the output tags -->
<CFOUTPUT QUERY="get_parcel_info">
<P>Parcel Number: #apn#<BR>
Owner: #owner#<BR>
Year Built: #yearblt#</P>
</CFOUTPUT>
</BODY>
</HTML>
```

Note that CFML tags are almost identical to those in the first example ("Listing File Contents"). The only change is to the <CFQUERY> tag, which uses a different SQL statement:

```
SELECT * FROM Parcel_Data WHERE APN IN (#PreserveSingleQuotes(OBJ_KEYS)#)
```

As with the previous example, the statement is selecting records from the Parcel_Data DSN. The difference is that the WHERE clause now points to a ColdFusion variable, #PreserveSingleQuotes(OBJ_KEYS)#. OBJ_KEYS refers to the parameter of the same name we specified in Autodesk MapGuide Author. As its name suggests, the PreserveSingleQuotes() function tells ColdFusion to

keep the single-quotes surrounding each feature key, instead of removing them automatically as it normally would.

This statement is basically saying "in Parcel_Data, select all records whose APN field matches OBJ_KEYS." Put more simply, it's saying "select the records that correspond to the selected features on the map." If OBJ_KEYS contains multiple keys, ColdFusion outputs the feature data associated with each key.

Creating an HTML Page to Display the Map

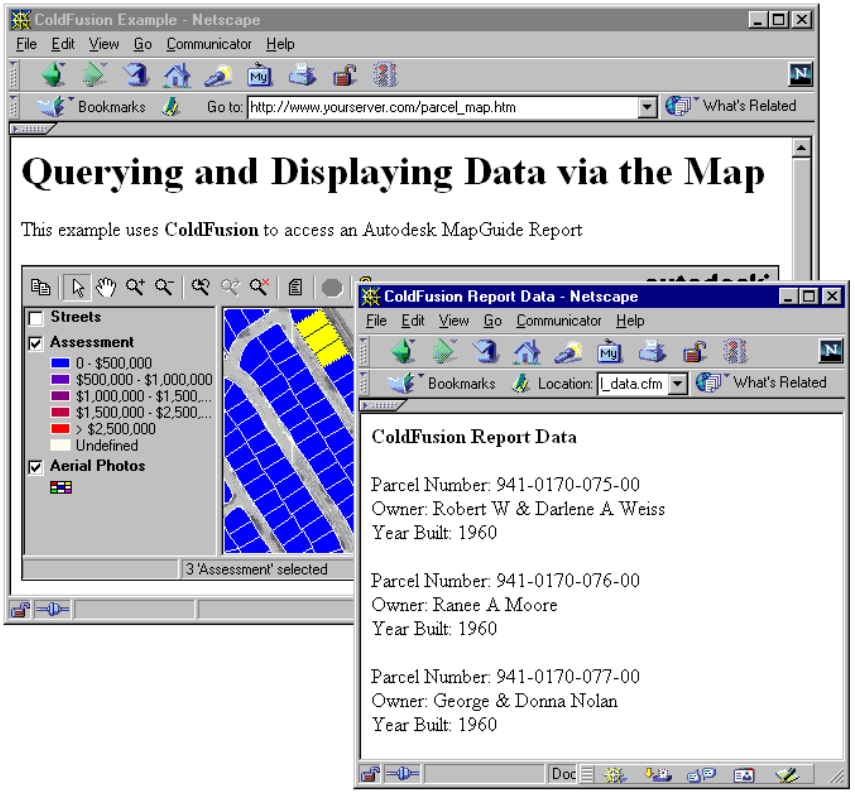
For more info... see "Displaying a Map in an HTML Page" in the *Autodesk MapGuide Viewer API Help*.

The last step is to create an HTML page to display our map. The following listing is for a file called *parcel_map.htm*.

```
<HTML>
<HEAD><TITLE>ColdFusion Example</TITLE></HEAD>
<BODY>
<H1>Querying and Displaying Data via the Map</H1>
<P>This example uses <b>ColdFusion</b> to access an Autodesk
MapGuide Report</P>
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL"
     VALUE="http://www.yourserver.com/maps/StarterApp.mwf>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf
     NAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Seeing the Results

Now we're ready to view *parcel_map.htm* in our web browser. Users can generate a report for one or more map features by selecting the features, right-clicking and choosing View ➤ Reports from the popup menu, and then selecting Parcel Data (CF).



Displaying the report in a new window

That looks pretty good, but we can still do a few things to improve the interface.

Redirecting Report Output

For more info... see "Viewer URL Parameters" in the *Autodesk MapGuide Viewer API Help*.

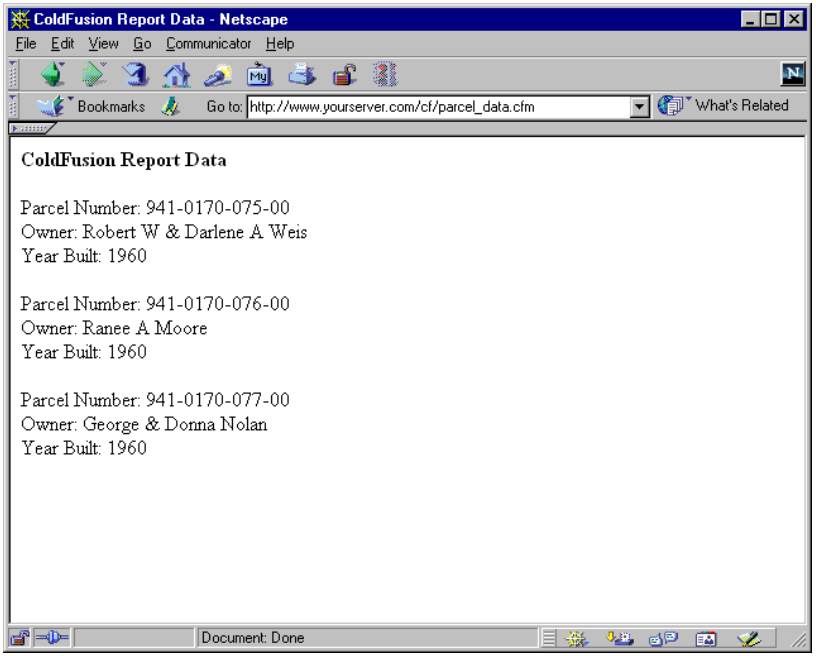
To avoid cluttering the desktop, let's generate the report in the current browser window, instead of displaying it in a new instance of the browser. Go back to *parcel_map.htm* and modify the embedded map code:

```
<OBJECT ID="myMap" width=600 height=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL"
     VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=_self>
   <EMBED src="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=_self
     NAME="myMap"WIDTH=600 HEIGHT=250>
</OBJECT>
```

Notice that we've added a Viewer URL parameter to the map reference:

```
http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=_self
```

ReportTarget specifies the window or frame in which you'd like your report to display. By specifying _self, we redirect the report output so that it displays in the current window.



Displaying the report in the current window

At first glance this appears to be a good solution, but it has some problems. Users might get confused about where they are. Worse yet, when they click the Back button, they will find that the map has been reloaded and the location they zoomed to has been lost. A better approach is to display the map and the report in two frames of the same window. Let's do that now.

Start by creating a standard HTML file that defines a frameset. The frameset should display the map on the left and a blank page on the right:

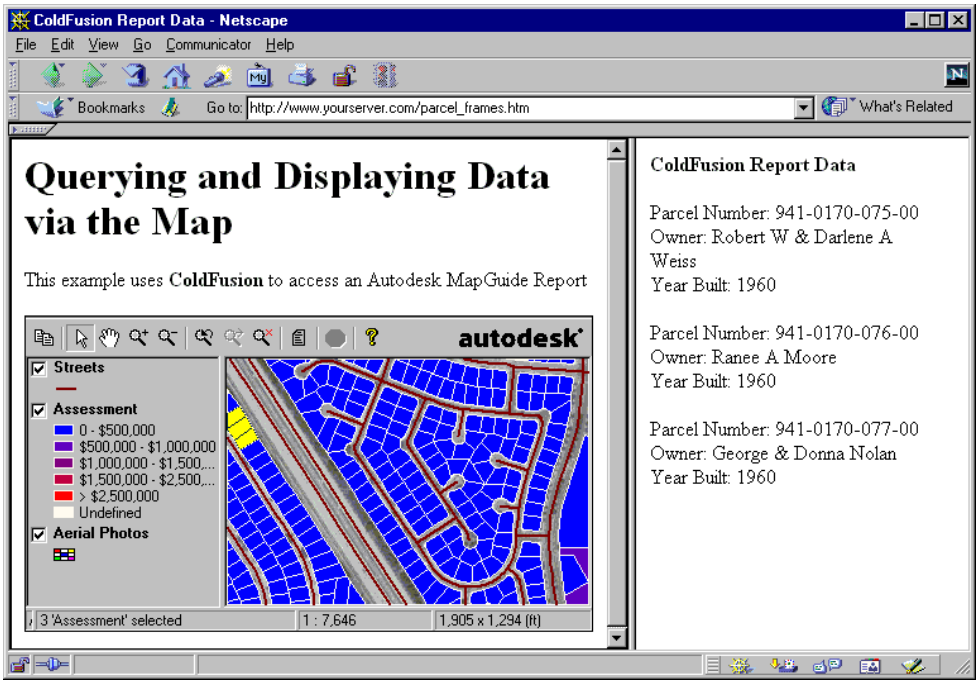
```
<HTML>
<HEAD>
   <TITLE>ColdFusion Report Data</TITLE>
</HEAD>
<!-- frames -->
<FRAMESET COLS="65%,*">
   <FRAME NAME="Left" SRC="parcel_map.htm" MARGINWIDTH="10"
      MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="yes">
   <FRAME NAME="Right" SRC="about:blank" MARGINWIDTH="10"
      MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="yes">
</FRAMESET>
</HTML>
```

Notice that we've assigned the names Left and Right to the frames. The source for Left is *parcel_map.htm*, the file containing our embedded map. The source for Right is about:blank, a standard browser function whose purpose is to display a blank window or frame.

Now that we have the frameset, let's go back to *parcel_map.htm* and change the ReportTarget parameter to Right, the name we assigned to our right-hand frame:

```
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
   CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
     App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
     portTarget=RightNAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
```

Let's see how it looks:



Displaying the report in a frame

Much better! Now your users can invoke as many reports as they want, without losing their place in the map or calling a new instance of the browser.

Adding a Button with the Viewer API

Now we'll make one last change to add some polish. An Autodesk MapGuide report is generated by right-clicking the map and then choosing View ➤ Reports from the popup menu. This interface is not immediately apparent to users, so we'll make it easier for them by creating a "Parcel Report" button that will display the report.

For more info...
See Chapter 2 and the *Autodesk MapGuide Viewer API Help* for more information on the Viewer API.

First we'll add the following <SCRIPT> tag to *parcel_map.htm*:

```
<SCRIPT>
   function getThisMap()
   {
      if (navigator.appName == "Netscape")
         return parent.Left.document.myMap;
      else
         return parent.Left.window.myMap;
   }

   function runReport()
   {
      parent.Right.document.write("<P>Select one or more parcels first.</P>");
      getThisMap().viewReport('Parcel Data (CF)');
   }
</SCRIPT>
```

The <SCRIPT> tag holds two JavaScript functions. The first function is our old friend getThisMap(), which we're using to smooth out some differences between Netscape and Internet Explorer (for more information, see "Accessing the Map Programmatically" on page 15). The second function, runReport(), displays our Autodesk MapGuide report.

The runReport() function consists of two statements. The first statement writes a line of text to the right-hand frame of our report application:

```
parent.Right.document.write("<P>Select one or more parcels first.</P>");
```

You'll notice that the text instructs users to select one or more map features. This instruction displays every time runReport() is invoked, regardless of whether the user has selected features. If features are selected, the instructions are replaced in the frame by the contents of the newly generated report; otherwise the instructions remain in the frame to provide feedback.

Note "parent" refers to the top-level frame and "Right" is the name we specified for our right-hand frame in *parcel_frames.htm*. Refer to third-party JavaScript documentation for more information on writing to frames and windows.

The second statement uses viewReport(), a Viewer API function, to run our report:

```
getThisMap().viewReport('Parcel Data (CF)');
```

The statement starts out by calling getThisMap(), which returns the map object in the web page. The map object is then passed to viewReport(), which directs Autodesk MapGuide to display a specified report, in this case "Parcel Data (CF)".

Now that our function is defined, we need a way to call it. Let's add a <FORM> element to *parcel_map.htm*:

```
<FORM>
   <INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()"</INPUT>
</FORM>
```

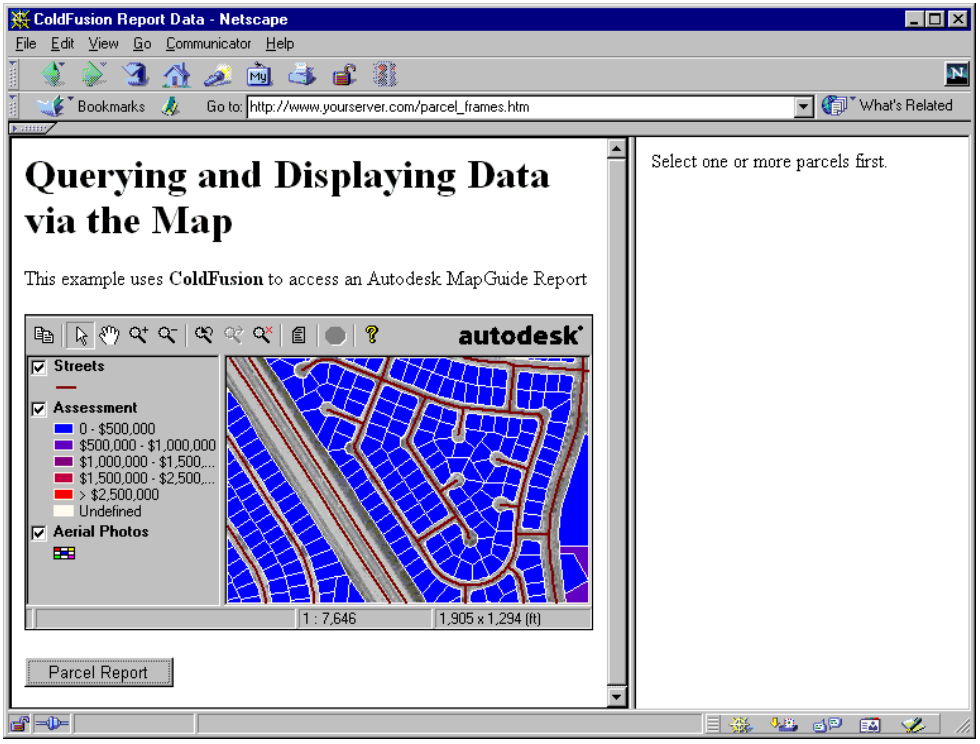
This is a standard HTML form consisting of a single button named "Parcel Report". By setting the value of ONCLICK to "runReport()", we specify that the function should be invoked each time a user clicks the button.

Note A JavaScript function must appear above the JavaScript code that calls it. This keeps users from trying to call a function before it has been parsed by the browser. JavaScript functions are typically defined in a single <SCRIPT> tag in the <HEAD> section of the HTML file.

Now let's look at the final text of *parcel_map.htm*, as well as the finished application. The illustration shows the results if no map features have been selected.

```
<HTML>
<HEAD>
   <TITLE>ColdFusion Example</TITLE>
   <!-- JavaScript functions -->
   <SCRIPT>
      // function #1
      function getThisMap()
      {
         if (navigator.appName == "Netscape")
            return parent.Left.document.myMap;
         else
            return parent.Left.window.myMap;
      }
      // function #2
      function runReport()
      {
         parent.Right.document.write("<P>Select one or more parcels first.</P>");
         getThisMap().viewReport('Parcel Data (CF)');
      }
   </SCRIPT>
</HEAD>

<BODY>
<H1>Querying and Displaying Data via the Map</H1>
<P>This example uses <b>ColdFusion</b> to access an Autodesk
MapGuide Report</P>
```

```
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
      App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
      portTarget=Right NAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
<!-- Parcel Report button -->
<FORM>
   <INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()"</INPUT>
</FORM>
</BODY>
</HTML>
```



The finished product

The next example shows you how to modify a database via the map.

Example—Modifying a Database via the Map

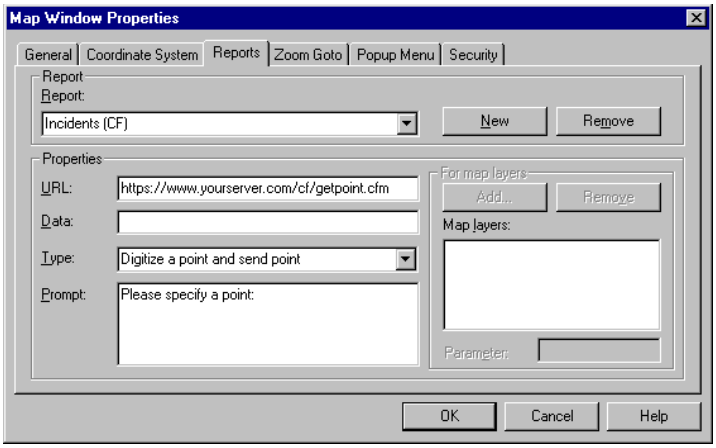
For more info...
For information on adding features to SDF files instead of to databases, see Chapter 5.

This example shows how to create an application that lets users add points to a map from their browsers. The points are stored in a database on the server and are visible to anyone else viewing the map. The example is of a hypothetical "Incident Log" application that will be used to track crimes and consists of the following components:

- Three ColdFusion files, *getpoint.cfm*, *showform.cfm*, and *insert.cfm*. As their names suggest, the files receive point coordinates from Autodesk MapGuide, display a form that takes additional user input, and add the point data to a database on the server.
- An Autodesk MapGuide report (and later, a custom menu item) that passes digitized point coordinates to *getpoint.cfm*.
- An HTML page to host the map (except for minor text changes, this page will be identical to *parcel_map.htm* from the previous example).
- An "Incidents" database table on the server and a new map layer (also called "Incidents") to display the contents of that table.

Setting Up the Report in Autodesk MapGuide Author

We'll start by specifying the report information in Autodesk MapGuide Author. Our report will be called "Incidents (CF)", and it will pass the lat/lon coordinates of a user-specified point to a CFM file whose URL is *http://www.yourserver.com/cf/getpoint.cfm*:



Dialog box specifications for "Incidents (CF)" report

Here are descriptions of how we used the options on the Reports tab:

Report	Specifies the name of the report as it appears in the Autodesk MapGuide Viewer. Our report is named "Incidents (CF)".
URL	Specifies the name and location of the report script, in this case *getpoint.cfm* on *www.yourserver.com*.
Data	We left this field blank but could have used it to pass additional URL parameters to *getpoint.cfm*.
Type	Specifies whether the report is based on the keys of selected features, or on the coordinates of a point the user clicks. We chose the second option, "Digitize a point and send point."
Prompt	Specifies the text to be displayed in a message box prompting users to specify a point. If this field is left blank, no message box is displayed.

When a user runs the Incidents (CF) report, Autodesk MapGuide prompts the user to specify a point. Then it invokes *getpoint.cfm*, passing the point's lat/lon coordinates as URL parameters. For example, if the user specified a point whose coordinate values were -121.943,37.721, the URL would look like this:

```
http://www.yourserver.com/getpoint.cfm?LAT=-121.943&LON=37.721
```

Creating the Report Scripts

Next we'll create the three CFM files: *getpoint.cfm*, *showform.cfm*, and *insert.cfm*.

The first file, *getpoint.cfm*, creates a small browser window and then calls a second file, *showform.cfm*, passing along the coordinate values it received from Autodesk MapGuide:

```
<SCRIPT LANGUAGE = "JavaScript">
window.close();
var loc = "showform.cfm?LAT=" + <CFOUTPUT>#LAT#</CFOUTPUT> +
   "&LON=" + <CFOUTPUT>#LON#</CFOUTPUT>;
win = window.open(loc,"ShowFormWin",
   "width=300,height=170,dependent=yes,resizable=yes");
win.focus();
</SCRIPT>
```

Note that *getpoint.cfm* consists of a single <SCRIPT> element containing a block of JavaScript code; because the file doesn't display any text, no other HTML tags are needed. Let's look at the code line by line.

When *getpoint.cfm* is first called it uses a default browser window similar to the one we saw in the previous example. The first line of code closes that window:

```
window.close();
```

Note Because the browser parses the entire <SCRIPT> block before running the first line of code, we can safely close the window, knowing our script will continue to run. Be aware, however, that this strategy will get you into trouble if your file contains function calls or other multiple <SCRIPT> blocks. See your JavaScript documentation for more information.

The next line constructs a URL and assigns it to a variable called "loc":

```
var loc = "showform.cfm?LAT=" + <CFOUTPUT>#LAT#</CFOUTPUT> +
   "&LON=" + <CFOUTPUT>#LON#</CFOUTPUT>;
```

Note that the line is a mix of both JavaScript code and ColdFusion <CFOUTPUT> tags. The <CFOUTPUT> tags contain two ColdFusion variables named #LAT# and #LON#. These variables are replaced on the server by the lat/lon values that Autodesk MapGuide provided, meaning the browser receives a line similar to the following:

```
var loc = "showform.cfm?LAT=" + "-121.943" + "&LON=" + "37.721";
```

The effect of this line is to create a variable called "loc" and to assign it the value showform.cfm?LAT=-121.943&LON=37.721.

The next line creates a new browser window, using the loc variable to supply the URL:

```
win = window.open(loc,"ShowFormWin",
   "width=300,height=170,dependent=yes,resizable=yes");
```

The last line shifts browser focus to the new window we just created:

```
win.focus();
```

Now, let's look at the second file, *showform.cfm*:

```
<HTML>
<HEAD>
   <TITLE>Attribute Input</TITLE>
</HEAD>
<BODY BGCOLOR="SILVER">
```

```
<CFOUTPUT>
<FORM Name=myForm METHOD="POST" ACTION="insert.cfm">
   <INPUT TYPE="hidden" NAME="rpt_lat" VALUE="#LAT#">
   <INPUT TYPE="hidden" NAME="rpt_lon" VALUE="#LON#">
   Incident Report:<BR>
   <INPUT TYPE="text" MAXLENGTH="30" NAME="rpt_info" SIZE="33"><BR>
   <BR>
   Reported By:<BR>
   <INPUT TYPE="text" MAXLENGTH="30" NAME="rpt_by" SIZE="33"><BR>
   <BR>
   <CENTER>
   <INPUT TYPE="submit" NAME="Submit" VALUE="OK">
   <INPUT TYPE="button" NAME="CancelButton" VALUE="Cancel"
      onClick="window.close()">
   </CENTER>
</FORM>
</CFOUTPUT>
</BODY>
</HTML>
```

The *showform.cfm* file does indeed show a form, which is used to enter a description of the crime (euphemistically referred to as an "incident").

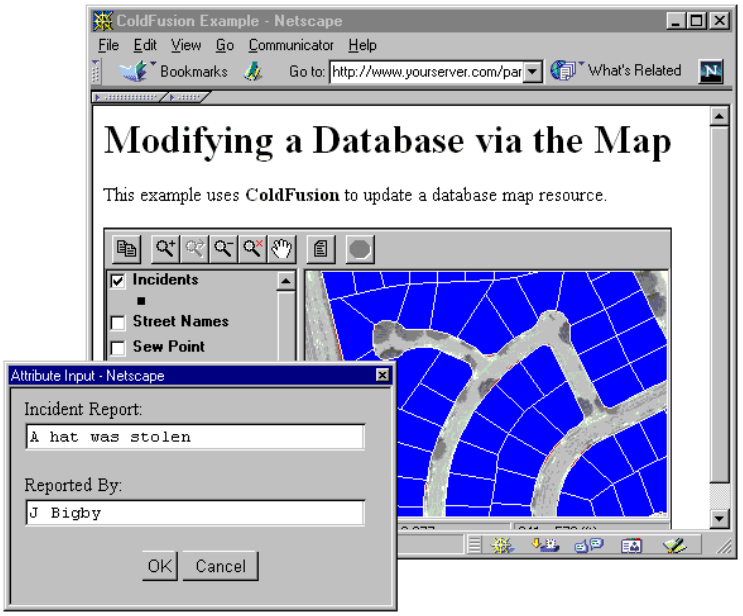
The bulk of the file is a standard HTML form. The form has been placed within <CFOUTPUT> tags to give us access to the ColdFusion variables #LAT# and #LON#. Once again, these variables are replaced on the server by the user-specified lat/lon coordinates.

For more info...
Refer to a third-party book on HTML for information about using forms.

In short, an HTML form collects data from the user and sends that data to a program in the form of a URL. The form in *showform.cfm* calls yet another CFM file, *insert.cfm*, passing it the following parameters:

- The latitude value represented by the ColdFusion variable #LAT#; this value is passed as the form parameter "rpt_lat".
- The longitude value represented by the ColdFusion variable #LON#; this value is passed as the form parameter "rpt_lon".
- An incident description entered in the form by a user; this description is passed as the form parameter "rpt_info".
- A name entered in the form by a user; this name is passed as the form parameter "rpt_by".

The following illustration shows the *showform.cfm* form, as displayed in the window created by *getpoint.cfm*.



Entering data into *showform.cfm*

Specifying the lat/lon points -121.943,37.721 by clicking the map and filling out the form as shown in the illustration will result in the following URL being constructed and passed to *insert.cfm*:

```
insert.cfm?rpt_lat=-121.943&rpt_lon=37.721&rpt_info=A+hat+was+stolen&rpt_by=J+Bigby
```

Now, let's see how *insert.cfm* handles the URL.

```
<CFQUERY NAME="InsertQuery" DATASOURCE="assessor">
INSERT into Incidents (lat, lon, description, reported_by)
values('#FORM.rpt_lat#','#FORM.rpt_lon#','#FORM.rpt_info#','#FORM.rpt_by#')
</CFQUERY>
<SCRIPT LANGUAGE = "JavaScript">
alert("Point added successfully! Reload the map to see your changes.");
window.close();
</SCRIPT>
```

The file consists of <CFQUERY> and <SCRIPT> tags. Like *getpoint.cfm*, the file contains no displayable text. The <CFQUERY> tag defines a query named "InsertQuery" using the "assessor" DSN from the previous examples. Note that the query name is defined but not used again in the file.

The <CFQUERY> element contains a single SQL statement, which is used to add the form data to the map-resource database:

```
INSERT into Incidents (lat, lon, description, reported_by)
values('#FORM.rpt_lat#','#FORM.rpt_lon#','#FORM.rpt_info#','#FORM.rpt_by#')
```

The SQL INSERT statement adds data to a database resource, in this case the Incidents table in the assessor DSN. The parenthetical values "lat", "lon", "description", and "reported_by" are the names of the database fields we want to supply values for. The parenthetical values #FORM.rpt_lat#, #FORM.rpt_lon#, #FORM.rpt_info#, and #FORM.rpt_by# represent the information we want to place into those fields, in this case the URL parameters passed form *showform.cfm*.

Now let's look at the contents of the <SCRIPT> element:

```
alert("Point added successfully! Reload the map to see your changes.");
window.close();
```

The first line displays an alert telling users to reload the map to see their changes. The second line closes the form window, leaving only the original map window.

Creating an HTML Page to Display the Map

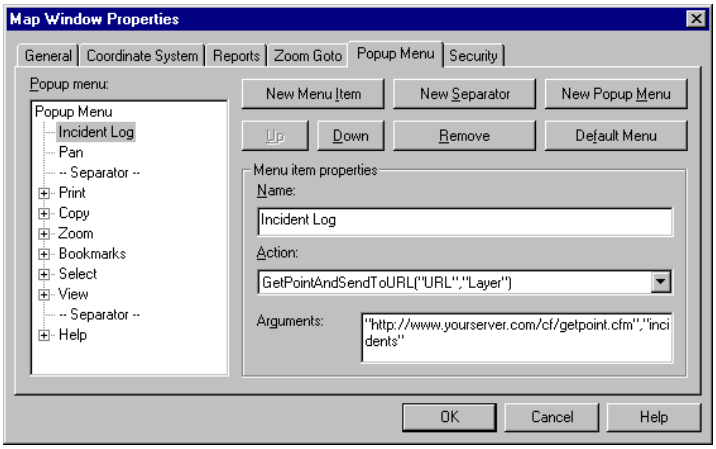
The last step is to create an HTML page to display our map. We'll use the *parcel_map.htm* file from the previous example, modifying the <H1> and the short paragraph of descriptive text, but leaving the rest of the file unchanged:

```
<HTML>
<HEAD>
   <TITLE>ColdFusion Example</TITLE>
</HEAD>
<BODY>
<!-- Only the next two lines are different -->
<H1>Modifying a Database via the Map</H1>
<P>This example uses <b>ColdFusion</b> to update a database map
resource</P>
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
       App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
       portTarget=Right NAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Creating a Custom Menu Item

As we designed it, our report has a few problems. The first is that it requires too many mouse clicks: users have to select View ➤ Reports from the popup menu, then they have to select Incidents (CF) from the list, then they have to clear the alert box that tells them to select a point, then they have to digitize the point. The other problem is that because our report isn't associated with a layer, users can add items to the Incidents map layer even when that layer isn't visible.

We can solve both of these problems by creating a custom menu item that takes the place of the report. We do so by selecting the following options on the Popup Menu tab of the Map Window Properties dialog box:



Dialog box specifications for 'Incident Log' menu item

Here are descriptions of how we used the options on the Popup Menu tab:

New Menu Item	Creates a new popup menu item below the item selected in the Popup Menu list.
Name	Specifies the name of the menu item as it will appear in the Viewer. Our menu item is named "Incident Log."
Action	Specifies the task to be performed by the menu item. We selected "GetPointAndSendToURL" from the drop-down list.
Arguments	Specifies arguments to use with the selected action, in this case the path to *getpoint.cfm* and the name of the layer we want to add data to.

When users select Incident Log from the popup menu, they will immediately be able to enter a point, thus bypassing several mouse clicks. Also, if the Incidents layer is not visible because the map is zoomed outside of the layer's display range, the Incidents Log menu item will be unavailable.

Accessing Your Application with the Viewer API

For more info...
Refer to the *Autodesk MapGuide Viewer API Help*.

Because the Incident Log application runs in a separate instance of the browser, it does not have programmatic access to the map window. This means the application cannot refresh the map automatically. (That's why we have a JavaScript alert() box telling the user to reload the map manually.)

To solve this problem and to avoid having the user need to reload the map manually, use the Viewer API to access the Incident Log application. Instead of creating a report or custom menu item, add a button or other interface element to the HTML page hosting the map (or to a frame or child window with access to that page). The button should invoke a JavaScript function that does the following:

- Uses the digitizePoint() method to get the coordinates of a user-specified point.
- Invokes *getpoint.cfm*, passing the point coordinates as URL parameters.
- Refreshes the map after *getpoint.cfm*, *showform.cfm*, and *insert.cfm* have completed their work.

This section described how to use ColdFusion to work with report scripts. The next section covers similar topics using ASP instead of ColdFusion.

Creating Report Scripts with ASP

ASP files are similar to ColdFusion templates: both are based on HTML, and both use a special extension to identify the file as one that requires special processing (not surprisingly, ASP uses *.asp*). Instead of tags, ASP files include scripts written in VBScript, a lightweight Visual Basic-like scripting language, or in JScript, Microsoft's version of JavaScript. VBScript is the more commonly used of the two languages.

While ColdFusion is designed specifically for web-database connectivity, ASP is a more general development environment. On one hand, this means you can do more with ASP than with ColdFusion. On the other hand, it takes longer to learn to do anything at all with ASP. Both products are excellent, but if you're a non-programmer, you'll probably be happier with ColdFusion.

Much of ASP's functionality is provided by *objects* and *components*. Objects and components are ActiveX DLLs, similar to those you would use with Microsoft Visual Basic. Objects are always available to VBScript; you do not have to explicitly create them to use them in your code. Components exist outside of ASP and must be created with ASP in order to be used. ASP also provides access to several server *events*; the *global.asa* file lets you add code for how those events should be handled.

Mostly, you will be working with the Server object, the Request object, and the Database Access component. To get a good idea of how ASP works, skim the descriptions below, and then look at the examples that follow.

Tip For more information on ASP, refer to the Microsoft Internet Information Server online documentation and the Microsoft web site (*www.microsoft.com*). ASP documentation is also available as part of the Microsoft Developer Network (MSDN) Library.

Summary of ASP Objects, Components, and Events

Summary of ASP objects

Object	Description
Application	Lets you create variables available to all users of an application.
Session	Lets you create variables that are available to one user at a time; session variables stay in memory as long as a user continues the session.
Request	Parses data submitted from the client to the server.
Response	Manages content returned to a browser by ASP.
Server	Provides a number of useful server methods, including CreateObject(), which you'll use to create a connection to your database map resources.

Summary of ASP components

Component	Description
Database Access	Reads and writes to OLE DB data sources.
File Access	Allows access to text files on your web site.
Browser Capabilities	Identifies the browser currently accessing the site and provides programmatic access to features the browser supports.
Ad Rotator	Controls the rotation of banner ads in a site.
Content Linking	Links separate web pages together so that users can scroll through them as a single page.

Note You can also create your own custom ActiveX components for ASP.

Summary of ASP events

Event	Description
Session_OnStart	Runs the first time a user accesses your application.
Session_OnEnd	Runs when a user's session times out or when a user quits your application.
Application_OnStart	Runs once, when the first page of your application is accessed for the first time by any user; does not run after another user accesses the first page of your application. The web server needs to be shut down for Application_OnStart to run again.
Application_OnEnd	Runs when the web server is shut down.

The following examples show how to create report scripts with ASP. We recommend that you read them in order.

Note For ColdFusion versions of the same examples, see "Creating Report Scripts with ColdFusion" on page 80.

Example—Listing File Contents with ASP

This example shows a simple server page that lists the contents of a map-resource database. Note that this page accesses the database directly, instead of using Autodesk MapGuide's reporting feature. Later, we'll see how Autodesk MapGuide fits into the picture.

You have an MWF file pointing to a database containing parcel information (lot number, street address, owner's name and so on) and you want to list the contents of the database at the bottom of an HTML page displaying the map. To do so, first rename the HTML file with an *.asp* extension and place it in a directory with script or execute permissions. Then add some code to specify the scripting language, establish a connection to the appropriate database records, and control the database output.

The next sections describe each of those tasks in more detail.

Specifying a Scripting Language

ASP scripts are written in VBScript, a lightweight Visual Basic-like scripting language, or in JScript, Microsoft's version of JavaScript. ASP files should begin with a line telling ASP which language you're using (although a default of VBScript is assumed if the line is omitted). Since we're using VBScript, our line will look like this:

```
<%@ LANGUAGE="VBSCRIPT"%>
```

Note the use of <% and %>, which identify the line as server-side code ASP should process.

Selecting Database Records

For more info...
Refer to the Autodesk MapGuide online help for more information on OLE DB.

Next, we'll add a some code to define a selected set of database records. This selection is known as a *recordset*. To come up with a recordset, we need to know which database table to connect to, and which records to select from that table. If your map links to a table called Parcel_Data through an OLE DB data source called Assessors, the recordset code will look like this:

```
<%
   Set dbConnection = Server.CreateObject("ADODB.Connection")
   dbConnection.Open("Assessor")
   SQLQuery = "SELECT * FROM Parcel_Data WHERE Year_Built = '1963'"
   Set RS = dbConnection.Execute(SQLQuery)
%>
```

This might seem complicated compared to ColdFusion's <CFQUERY> tag, but it will look familiar to Visual Basic programmers. The end result is a Recordset object variable, RS, which represents all houses in Parcel_Data that have a Year_Built value of "1963".

Note Don't be put off by this code if you are unfamiliar with Visual Basic. All of your ASP database queries will follow this basic format, with only the DSN and SQL statement varying.

Let's go through the recordset script line by line. The first line of code uses the CreateObject method of the Server object to create a new Connection object, which is assigned to a variable called dbConnection.

```
Set dbConnection = Server.CreateObject("ADODB.Connection")
```

The next line opens a connection to the data source name (DSN), in this case "Assessor", and assigns that connection to the dbConnection variable. Note that Open is a method of the Connection object, in this case dbConnection.

```
dbConnection.Open("Assessor")
```

The third line creates a variable that holds a SQL statement specifying the database records we want to work with. As we saw in the previous example, selecting everything in the database produces a page of browser-choking proportions, so our SQL query is limited to houses built in 1963.

```
SQLQuery = "SELECT * FROM Parcel_Data WHERE Year_Built = '1963'"
```

The last line puts it all together, creating a Recordset object and assigning it to an object variable named RS. Note that Execute is a method of the Connection object, in this case dbConnection. We're using Execute to run the SQL statement we assigned to SQLQuery.

```
Set RS = dbConnection.Execute(SQLQuery)
```

Controlling the Output

Now that we have our Recordset object, let's add a block of code that controls how the database output is displayed on the page. This code should appear within <BODY>, at the location where you want the database output to appear. If you want to display the parcel number, owner's name, and year built, your output code will look like this:

```
<%
   Do While Not RS.EOF
%>
<P>Parcel Number: <%=RS("APN")%><BR>
Owner: <%=RS("Owner_Name")%><BR>
Year Built: <%=RS("Year_Built")%></P>
<%
   RS.MoveNext
   Loop
%>
```

If you're used to client-side scripting, this code might look peculiar. Notice how it is actually two different script tags that operate on HTML code sandwiched in the middle.

Let's look at the HTML portion first:

```
<P>Parcel Number: <%=RS("APN")%><BR>
Owner: <%=RS("Owner_Name")%><BR>
Year Built: <%=RS("Year_Built")%></P>
```

As with ColdFusion, this is standard HTML plus a few variables. ASP variables use the standard ASP script tags (<% and %>), as well as an equal sign that tells ASP to substitute the actual value for the variable. In this case, the value is a field in your map-resource database. For example, RS("APN") is the APN column in the database represented by the RS object you created earlier.

Without the accompanying script tags, the HTML would display the APN, Owner_Name, and Year_Built fields for only the first record in the database:

Parcel Number: 941-0103-003-01
Owner: James P & Bonnie G Reed
Year Built: 1963

This is a good start, but not quite what we want. To cycle through the records, we'll need to add some sort of looping code. That's what the two scripts are for.

The beginning script contains a single line, which operates on the RS object. RS.EOF represents RS object's "end-of-file" property. In effect, the line is saying "do the following until you reach the end."

```
Do While Not RS.EOF
```

The ending block contains two lines, one that advances to the next record in the recordset and the another that finishes up the loop structure:

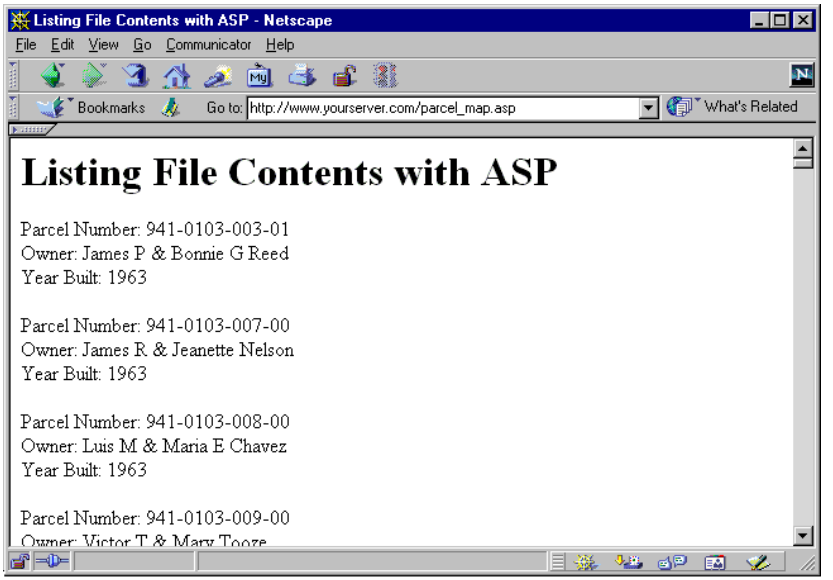
```
RS.MoveNext
Loop
```

Now we're ready to load the page in our browser.

Seeing the Results

Here's a listing of the complete ASP file *parcel_report.asp*, followed by an illustration of the page as it appears in a browser:

```
<HTML>
<HEAD>
<!-- code to create recordset -->
<%
   Set dbConnection = Server.CreateObject("ADODB.Connection")
   dbConnection.Open ("Assessor")
   SQLQuery = "SELECT * FROM Parcel_Data WHERE YearBlt = '1963'"
   Set RS = dbConnection.Execute(SQLQuery)
%>
<TITLE>ASP Test #1</TITLE>
</HEAD>
<BODY>
<H1>ASP Test #1</H1>
<!-- output code -->
<%
   Do While Not RS.EOF
%>
<P>Parcel Number: <%=RS("APN")%><BR>
Owner: <%=RS("Owner")%><BR>
Year Built: <%=RS("yearblt")%></P>
<%
   RS.MoveNext
   Loop
%>
</BODY>
</HTML>
```



The HTML output

For more info...
Refer to the Microsoft Internet Information Server online documentation and the Microsoft web site (*www.microsoft.com*).

Like the earlier ColdFusion example, this page is very simple, only hinting at the power of what you can do with ASP. And like the earlier example, this is not really an Autodesk MapGuide application. The database happens to be an Autodesk MapGuide resource, but it could be any database you have access to through a DSN. In most cases, you will want to access your databases through the Autodesk MapGuide Viewer by linking them to features and layers in the map. The examples that follow show you how to do this.

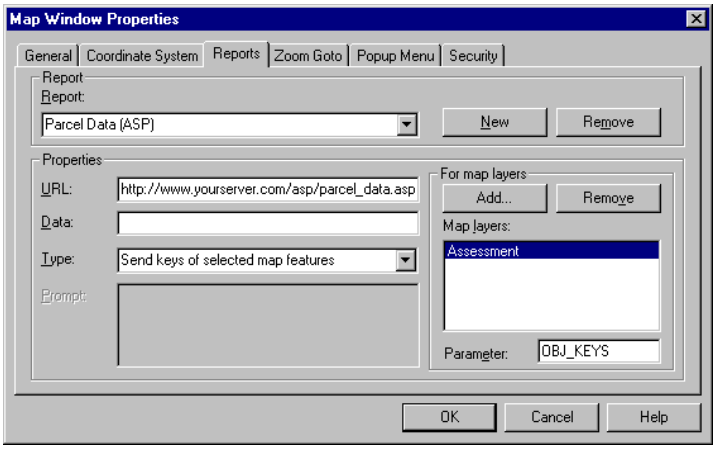
Example—Querying and Displaying Data via the Map

Now that we've seen how ASP works, let's use it with Autodesk MapGuide.

Setting Up the Report in Autodesk MapGuide Author

We'll start by specifying report information in Autodesk MapGuide Author.

Our report will be called "Parcel Data (ASP)", and it will access a server page whose URL is *http://www.yourserver.com/asp/parcel_report.asp*. We want the server page to display information about user-selected features on a layer called "Assessment." The Map Window Properties dialog box shown next reflects our selection.



Dialog box specifications for 'Parcel Data (ASP)'

Here are descriptions of how we used the options on the Reports tab:

Report	Specifies the name of the report as it appears in the Autodesk MapGuide Viewer. Our report is named "Parcel Data (ASP)".
URL	Specifies the name and location of the report script, in this case *parcel_report.asp* on *www.yourserver.com*.
Data	We left this field blank but could have used it to pass additional URL parameters to *parcel_report.asp*. For example, if our server page contained definitions for more than one query, we might have passed a parameter telling the file which of the queries to run.
Type	Specifies whether the report is based on the keys of selected features (as this one is), or on the coordinates of a point the user clicks.
For Map Layers	Specifies the layer or layers you want the report to be linked to. Our report operates only on features on the Assessment layer.
Parameter	Specifies the name of the URL parameter used to send the feature key (or keys) to *parcel_report.asp*. The name can be anything you want, as long as it matches the name you specified in *parcel_report.asp*. We've selected the Autodesk MapGuide Author default, "OBJ_KEYS".

When a user selects one or more features from the Assessment layer and runs the Parcel Data (ASP) report, Autodesk MapGuide constructs a URL that invokes *parcel_report.asp* and tells it to generate a report on the selected features, which are identified by their OBJ_KEY values. If the user selected a single feature whose key was "941-0176-003-00", the URL would look like this:

```
http://www.yourserver.com/asp/parcel_report.asp?OBJ_KEYS='941-0176-
003-00'
```

If the user selected multiple features, the URL might look like this:

```
http://www.yourserver.com/asp
  parcel_report.asp?OBJ_KEYS='941-0176-003-00','941-0176-006-00','941-0176-004-00'
```

Note that OBJ_KEYS is represented as a standard URL parameter. To ASP, this parameter is no different from one submitted by an HTML form element. As we'll see in the next section, ASP processes it accordingly.

Creating the Report Script

Now let's create the ASP file that will process the Autodesk MapGuide report. The following listing is for the *parcel_report.asp* file:

```
<HTML>
<HEAD>
   <TITLE>ASP Report Data</TITLE>
</HEAD>
<BODY>
<!-- code to create recordset -->
<%
Set dbConnection = Server.CreateObject("ADODB.Connection")
dbConnection.Open ("assessor")
SQLQuery = "SELECT * FROM Parcel_Data WHERE APN IN (" & request.form ("OBJ_KEYS") & ")"
Set RS = dbConnection.Execute(SQLQuery)
%>
<H1>ASP Report Data</H1>
<!-- output code -->
<%
   Do While Not RS.EOF
%>
<P>Parcel Number: <%=RS("APN")%><BR>
Owner: <%=RS("Owner")%><BR>
Year Built: <%=RS("yearblt")%></P>
<%
   RS.MoveNext
   Loop
%>
</BODY>
</HTML>
```

Note that the VBScript code is almost identical to that in the first example ("Example—Listing File Contents with ASP" on page 104). The only change is to the value we assign the SQLQuery variable:

```
SQLQuery = "SELECT * FROM Parcel_Data WHERE APN IN (" & request.form ("OBJ_KEYS") & ")"
```

As with the previous example, the statement is selecting records from the Parcel_Data DSN. The difference is that the WHERE clause now points to Request.Form, the ASP Request object's Form collection. The Request object is used by ASP to parse submitted data received from a client as part of a URL. Form is a collection representing the URL parameters, which can be accessed from the collection by name. In this case, the collection has only one member, the OBJ_KEYS parameter we specified in Autodesk MapGuide Author.

The SQL statement is basically saying "in Parcel_Data, select all records whose APN field matches OBJ_KEYS." Put more simply, it's saying "select the records that correspond to the selected features on the map." If OBJ_KEYS contains multiple keys, ASP outputs the feature data associated with each key.

Creating an HTML Page to Display the Map

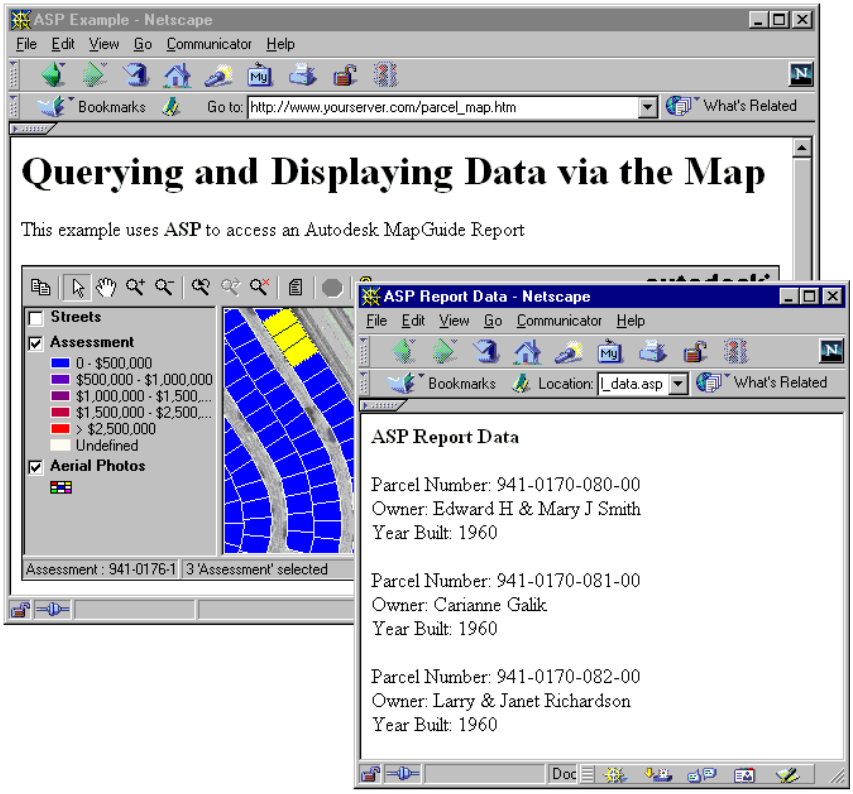
For more info...
see "Displaying a Map in an HTML Page" in the *Autodesk MapGuide Viewer API Help*.

The last step is to create an HTML page to display our map. The following listing is for a file called *parcel_map.htm*.

```
<HTML>
<HEAD>
   <TITLE>ASP Example</TITLE>
</HEAD>
<BODY>
<H1>Querying and Displaying Data via the Map</H1>
<P>This example uses <b>ASP</b> to access an Autodesk MapGuide Re-
port</P>
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL"
      VALUE="http://www.yourserver.com/maps/StarterApp.mwf>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf
      NAME="myMap"
      WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Seeing the Results

Now we're ready to view *parcel_map.htm* in our web browser. Users can generate a report on one or more map features by selecting the features, selecting View ➤ Reports from the popup menu, and then selecting Parcel Data (ASP).



Displaying the report in a new window

That looks pretty good, but we can still do a few things to improve the interface.

Redirecting Report Output

For more info...
see "Viewer URL Parameters," in the *Autodesk MapGuide Viewer API Help*.

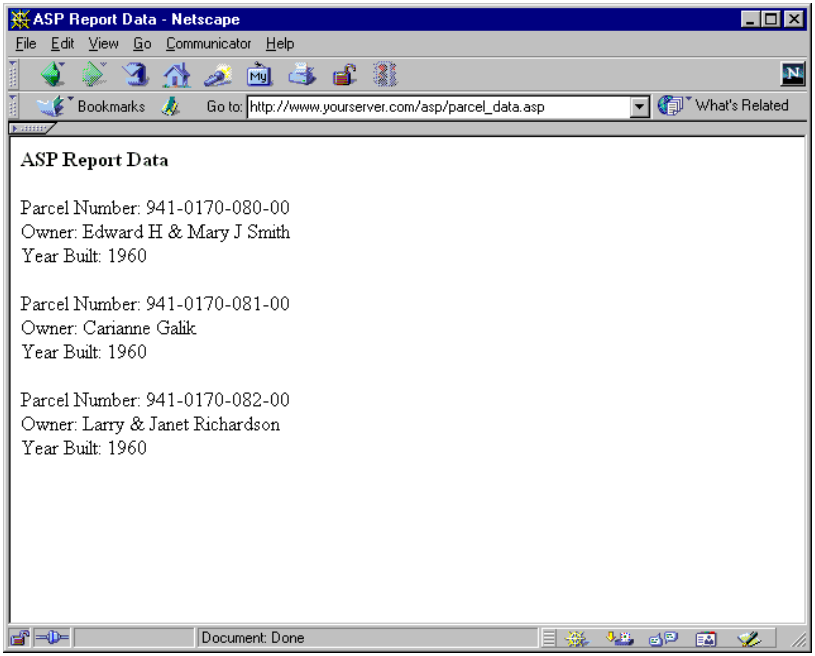
To avoid cluttering the desktop, let's generate the report in the current browser window, instead of displaying it in a new instance of the browser. Go back to *parcel_map.htm* and modify the embedded map code:

```
<OBJECT ID="myMap" width=600 height=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL"
     VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTar-
get=_self>
   <EMBED src="http://www.yourserver.com/maps/StarterApp.mwf?ReportTar-
get=_self
     NAME="myMap"WIDTH=600 HEIGHT=250>
</OBJECT>
```

Notice that we've added a Viewer URL parameter to the map reference:

```
http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=_self
```

ReportTarget specifies the window or frame in which you'd like your report to display. By specifying _self, we redirect the report output so that it displays in the current window.



Displaying the report in the current window

At first glance this appears to be a good solution, but it has some problems. Users might get confused about where they are. Worse yet, when they click the Back button, they will find that the map has been reloaded and the location they zoomed to has been lost. A better approach is to display the map and the report in two frames of the same window. Let's do that now.

Start by creating a standard HTML file that defines a frameset. The frameset should display the map on the left and a blank page on the right:

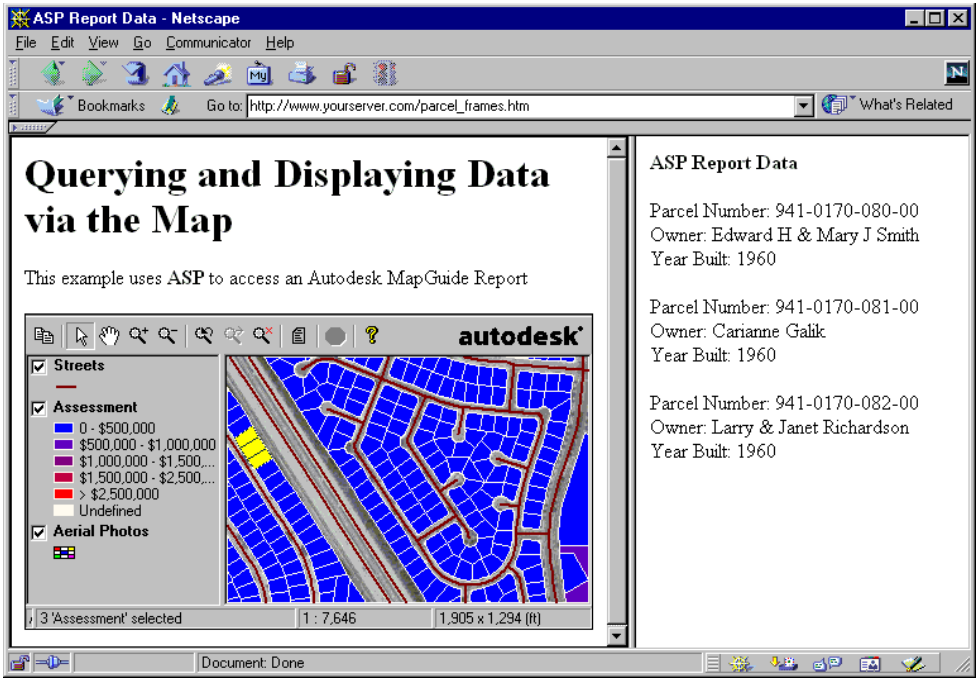
```
<HTML>
<HEAD>
   <TITLE>ASP Report Data</TITLE>
</HEAD>
<!-- frames -->
<FRAMESET COLS="65%,*">
   <FRAME NAME="Left" SRC="parcel_map.htm" MARGINWIDTH="10"
      MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="yes">
   <FRAME NAME="Right" SRC="about:blank" MARGINWIDTH="10"
      MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="yes">
</FRAMESET>
</HTML>
```

Notice that we've assigned the names Left and Right to the frames. The source for Left is *parcel_map.htm*, the file containing our embedded map. The source for Right is about:blank, a standard browser function whose purpose is to display a blank window or frame.

We have the frameset, so let's go back to *parcel_map.htm* and change the ReportTarget parameter to Right, the name we assigned to our right-hand frame:

```
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
   CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
       App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
       portTarget=RightNAME="myMap"WIDTH=600 HEIGHT=250>
</OBJECT>
```

Now let's see how it looks.



Displaying the report in a frame

This is an improvement because your users can now invoke as many reports as they want, without losing their place in the map or calling a new instance of the browser.

Adding a Button with the Viewer API

Now we'll make one last change to add some polish. An Autodesk MapGuide report is generated by right-clicking the map and then choosing View ➤ Reports from the popup menu. This interface is not immediately apparent to users, so we'll make it easier for them by creating a "Parcel Report" button that will display the report.

For more info...
See Chapter 2 and the *Autodesk MapGuide Viewer API Help* for more information on the Viewer API.

First we'll add the following <SCRIPT> tag to *parcel_map.htm*:

```
<SCRIPT>
   function getThisMap()
   {
      if (navigator.appName == "Netscape")
         return parent.Left.document.myMap;
      else
         return parent.Left.window.myMap;
   }

   function runReport()
   {
      parent.Right.document.write("<P>Select one or more parcels first.</P>");
      getThisMap().viewReport('Parcel Data (ASP)');
   }
</SCRIPT>
```

The <SCRIPT> tag holds two JavaScript functions. The first function is our old friend getThisMap(), which we're using to smooth out some differences between Netscape and Internet Explorer (for more information, see "Accessing the Map Programmatically" on page 15). The second function, runReport(), displays our Autodesk MapGuide report.

The runReport() function consists of two statements. The first statement writes a line of text to the right-hand frame of our report application:

```
parent.Right.document.write("<P>Select one or more parcels first.</P>");
```

You'll notice that the text instructs users to select one or more map features. This instruction displays every time runReport() is invoked, regardless of whether the user has selected features. If features are selected, the instructions are replaced in the frame by the contents of the newly generated report; otherwise, the instructions remain in the frame to provide feedback.

Note "parent" refers to the top-level frame and "Right" is the name we specified for our right-hand frame in *parcel_frames.htm*. Refer to a third-party JavaScript manual for more information on writing to frames and windows.

The second statement uses viewReport(), a Viewer API function, to run our report:

```
getThisMap().viewReport('Parcel Data (ASP)');
```

The statement starts out by calling getThisMap(), which returns the appropriate map feature. That feature is then passed to viewReport(), which directs Autodesk MapGuide to display a specified report, in this case "Parcel Data (ASP)."

Now that our function is defined, we need a way to call it. Let's add a <FORM> element to *parcel_map.htm*:

```
<FORM>
   <INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()"</INPUT>
</FORM>
```

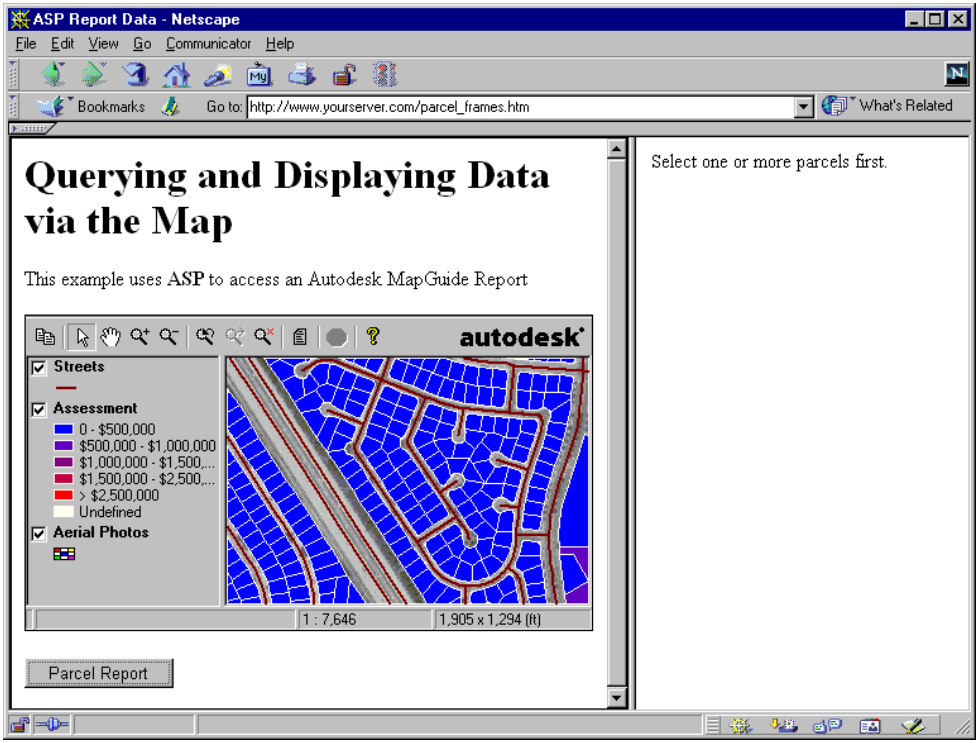
This is a standard HTML form consisting of a single button named "Parcel Report". By setting the value of ONCLICK to "runReport()", we specify that the function should be invoked each time a user clicks the button.

Note A JavaScript function must appear above the JavaScript code that calls it. This keeps users from trying to call a function before it has been parsed by the browser. JavaScript functions are typically defined in a single <SCRIPT> tag in the <HEAD> section of the HTML file.

Now let's look at the final text of *parcel_map.htm*, as well as the finished application. The illustration shows the results if no map features have been selected.

```
<HTML>
<HEAD>
   <TITLE>ASP Example</TITLE>
   <!-- JavaScript functions -->
   <SCRIPT>
      // function #1
      function getThisMap()
      {
         if (navigator.appName == "Netscape")
            return parent.Left.document.myMap;
         else
            return parent.Left.window.myMap;
      }
      // function #2
      function runReport()
      {
         parent.Right.document.write("<P>Select one or more parcels first.</P>");
         getThisMap().viewReport('Parcel Data (ASP)');
      }
   </SCRIPT>
</HEAD>
<BODY>
<H1>Querying and Displaying Data via the Map</H1>
<P>This example uses <b>ASP</b> to access an Autodesk MapGuide Re-
port</P>
```

```
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
       App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
       portTarget=RightNAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
<!-- Parcel Report button -->
<FORM>
   <INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()"</INPUT>
</FORM>
</BODY>
</HTML>
```



The finished product

The next example shows you how to modify a database via the map.

Example—Modifying a Database via the Map

For more info...
For information on adding features to SDF files instead of to databases, see Chapter 5.

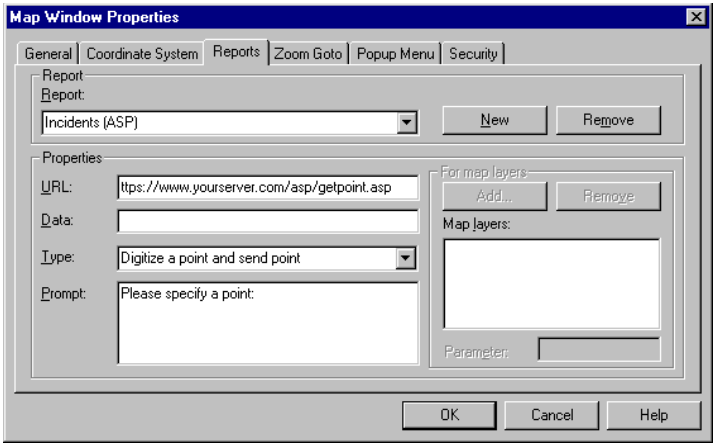
This example shows how to create an application that lets users add points to a map from their browsers. The points are stored in a database on the server and are visible to anyone else viewing the map.

The example is of a hypothetical "Incident Log" application that will be used to track crimes. The application consists of the following components:

- Three server pages, *getpoint.asp*, *showform.asp*, and *insert.asp*. As their names suggest, the files receive point coordinates from Autodesk MapGuide, display a form that takes additional user input, and add the point data to a database on the server.
- An Autodesk MapGuide report (and later, a custom menu item) that passes digitized point coordinates to *getpoint.asp*.
- An HTML page to host the map (except for minor text changes, this page will be identical to *parcel_map.htm* from the previous example).
- An "Incidents" database table on the server and a new map layer (also called "Incidents") to display the contents of that table.

Setting Up the Report in Autodesk MapGuide Author

We'll start by specifying the report information in Autodesk MapGuide Author. Our report will be called "Incidents (ASP)", and it will pass the lat/lon coordinates of a user-specified point to server page whose URL is *http://www.yourserver.com/asp/getpoints.asp*:



Dialog box specifications for 'Incidents (ASP)' report

Here are descriptions of how we used the options on the Reports tab:

Report	Specifies the name of the report as it appears in the Autodesk MapGuide Viewer. Our report is named "Incidents (ASP)".
URL	Specifies the name and location of the report script, in this case *getpoint.asp* on *www.yourserver.com*.
Data	We left this field blank but could have used it to pass additional URL parameters to *getpoint.asp*.
Type	Specifies whether the report is based on the keys of selected features, or on the coordinates of a point the user clicks. We chose the second option, "Digitize a point and send point."
Prompt	Specifies the text to be displayed in a message box prompting users to specify a point. If this field is left blank, no message box is displayed.

When a user runs the Incidents (ASP) report, Autodesk MapGuide prompts the user to specify a point. Then it invokes *getpoint.asp*, passing the point's lat/lon coordinates as URL parameters (more specifically, the coordinates are sent as form data via the HTTP POST method). For example, if the user specified a point whose coordinate values were -121.943,37.721, the URL would look like this:

```
http://www.yourserver.com/getpoint.asp?LAT=-121.943&LON=37.721
```

Creating the Report Scripts

Next we'll create the files *getpoint.asp*, *showform.asp*, and *insert.asp*.

The first file, *getpoint.asp*, creates a small browser window and then calls a second file, *showform.asp*, passing along the coordinate values it received from Autodesk MapGuide. Here is the first file, *getpoint.asp*:

```
<SCRIPT language="JavaScript">
window.close( );
var loc = "showform.asp?LAT=" + "<%=Request.Form("lat")%>"
   + "&LON=" + "<%=Request.Form("lon")%>";
win = window.open(loc,"ShowFormWin",
   "width=300,height=170,dependent=yes,resizable=yes");
win.focus();
</SCRIPT>
```

Note that *getpoint.asp* consists of a single <SCRIPT> element containing a block of JavaScript code; because the file doesn't display any text, no other HTML tags are needed. Let's look at the code line by line.

When *getpoint.asp* is first called it uses a default browser window similar to the one we saw in the previous example. The first line of code closes that window:

```
window.close();
```

Note Because the browser parses the entire <SCRIPT> block before running the first line of code, we can safely close the window, knowing our script will continue to run. Be aware, however, that this strategy will get you into trouble if your file contains function calls or other multiple <SCRIPT> blocks. See your JavaScript documentation for more information.

The next line constructs a URL and assigns it to a variable called "loc":

```
var loc = "showform.asp?LAT=" + "<%=Request.Form("lat")%>"
   + "&LON=" + "<%=Request.Form("lon")%>";
```

Note that the line is a mix of both JavaScript and ASP code. The ASP code is processed first, on the server. Then the line is sent to the browser as standard JavaScript. Let's look at how it works.

As you might recall from the previous example, ASP variables use the standard ASP script tags (<% and %>), as well as an equal sign that tells ASP to substitute the actual value for the variable. In this case, the variables hold the values `Request.Form("lat")` and `Request.Form("lon")`, both of which refer to Request.Form, the ASP Request object's Form collection. The Request object is used by ASP to parse submitted data received from a client as part of a URL. Form is a collection representing HTML form parameters transmitted via the HTTP POST method; these parameters can be accessed from the collection by name. In this case, the collection has two members: the LAT and LON parameters that were posted to the file by the Autodesk MapGuide Viewer. After the ASP code is processed, a line similar to the following is sent to the browser:

```
var loc = "showform.asp?LAT=" + "-121.943" + "&LON=" + "37.721";
```

The effect of this line is to create a variable called "loc" and to assign it the value showform.asp?LAT=-121.943&LON=37.721.

The next line creates a new browser window, using the loc variable to supply the URL:

```
win = window.open(loc,"ShowFormWin",
   "width=300,height=170,dependent=yes,resizable=yes");
```

The last line shifts browser focus to the new window we just created:

```
win.focus();
```

Now, let's look at the second file, *showform.asp*:

```
<HTML>
<HEAD>
   <TITLE>Attribute Input</TITLE>
</HEAD>
<BODY BGCOLOR="SILVER">
<FORM Name=myForm METHOD="POST" ACTION="insert.asp">
   <INPUT TYPE="HIDDEN" NAME="rpt_lat"
      VALUE="<%=Request.QueryString("lat")%>">
   <INPUT TYPE="HIDDEN" NAME="rpt_lon"
      VALUE="<%=Request.QueryString("lon")%>">
   Incident Report:<BR>
   <INPUT TYPE="TEXT" MAXLENGTH="30" NAME="rpt_info" SIZE="33"><BR>
   <BR>
   Reported By:<BR>
   <INPUT TYPE="TEXT" MAXLENGTH="30" NAME="rpt_by" SIZE="33"><BR>
   <BR>
   <CENTER>
   <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="OK">
   <INPUT TYPE="button" NAME="CancelButton" VALUE="Cancel"
      onClick="window.close()">
   </CENTER>
</FORM>
</BODY>
</HTML>
```

The *showform.asp* file does indeed show a form, which is used to enter a description of the crime (euphemistically referred to as an "incident").

The form follows HTML syntax, but also contains the ASP variables `<%=Request.QueryString("lat")%>` and `<%=Request.QueryString("lon")%>`. The Request.QueryString collection is similar to Request.Form, but instead of holding HTML form values transmitted via the HTTP GET method, it can hold either of the following:

- HTML form parameters transmitted via the HTTP POST method
- URL parameters that were added to the URL directly, instead of being generated by a form

We use QueryString in this case, because the URL parameters that were sent to *showform.asp* were created explicitly by JavaScript code in *getpoint.asp*.

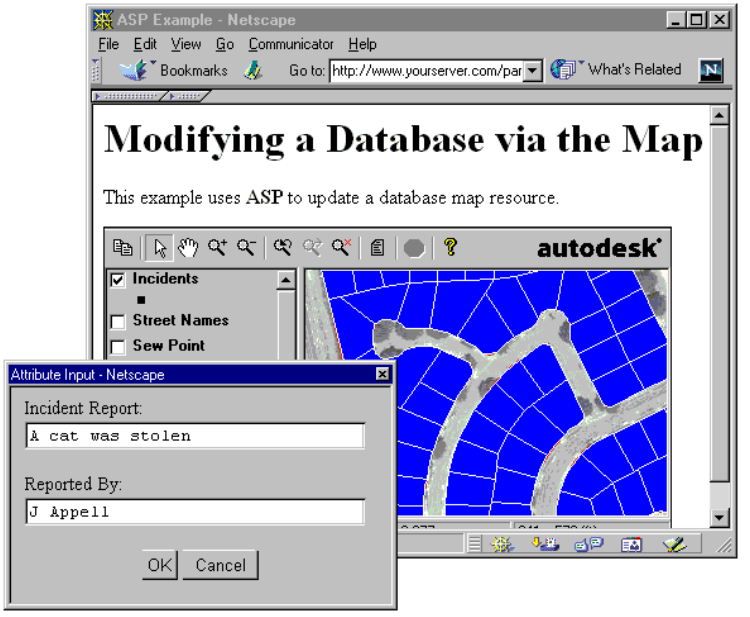
Note Use Request.Form if your data is being transmitted from an HTML form via the HTTP POST method. Use Request.QueryString if your data is being transmitted from an HTML form via the HTTP GET method, or if it is coming from a URL parameter not associated with any form.

For more info...
Refer to a third-party book on HTML for information about using forms.

In short, an HTML form collects data from the user and sends that data to a program in the form of a URL. The form in *showform.asp* calls yet another ASP file, *insert.asp*, passing it the following parameters:

- The latitude value obtained from the ASP variable <%=Request.QueryString("lat")%>; this value is passed as the form parameter "rpt_lat".
- The longitude value obtained from the ASP variable <%=Request.QueryString("lon")%>; this value is passed as the form parameter "rpt_lon".
- An incident description entered in the form by a user; this description is passed as the form parameter "rpt_info".
- A name entered in the form by a user; this name is passed as the form parameter "rpt_by".

The following illustration shows the *showform.asp* form, as displayed in the window created by *getpoint.asp*:



Entering data into the *showform.asp* form

Specifying the lat/lon points -121.943,37.721 by clicking the map and filling out the form as shown in the illustration will result in the following URL being constructed and passed to *insert.asp*:

```
insert.asp?rpt_lat=-121.943&rpt_lon=37.721&rpt_info=A+cat+was+stolen&rpt_by=J+Appell
```

Now we'll see how *insert.asp* handles the URL:

```
<%
   Set dbConnection = Server.CreateObject("ADODB.Connection")
   dbConnection.Open("assessor")
   SQLQuery =
      "INSERT into Incidents (lat, lon, description, reported_by)" & _
      "values('" & Request.Form("rpt_lat") & "','" & _
      Request.Form("rpt_lon") & "','" & Request.Form("rpt_info") & _
      "','" & Request.Form("rpt_by") & "')"
   dbConnection.Execute(SQLQuery)
%>
<SCRIPT language="JavaScript">
alert("Point added successfully! Reload the map to see your changes.");
window.close();
</SCRIPT>
```

Like *getpoint.asp*, the file contains no displayable text. Instead, it contains two blocks of code. One is an ASP script, written in VBScript. The other is an HTML <SCRIPT> element containing JavaScript code.

Let's go through the ASP code line by line. The first line of code uses the CreateObject method of the Server object to create a new Connection object, which is assigned to a variable called dbConnection.

```
Set dbConnection = Server.CreateObject("ADODB.Connection")
```

The next line opens a connection to the data source name (DSN), in this case "Assessor," and assigns that connection to the dbConnection variable. Note that Open is a method of the Connection object, in this case dbConnection.

```
dbConnection.Open("Assessor")
```

The third line creates a variable that holds an SQL statement specifying the data we want to add to the Incidents table:

```
   SQLQuery =
      "INSERT into Incidents (lat, lon, description, reported_by)" & _
      "values('" & Request.Form("rpt_lat") & "','" & _
      Request.Form("rpt_lon") & "','" & Request.Form("rpt_info") & _
      "','" & Request.Form("rpt_by") & "')"
```

Because *insert.asp* is receiving its information directly from a form (instead of from a URL we constructed programmatically), we call Request.Form instead of Request.QueryString. After Request.Form supplies the values from *showform.asp*, the line looks like this:

```
   SQLQuery =
      "INSERT into Incidents (lat, lon, description, reported_by)" & _
      "values('-121.943','37.721','A cat was stolen','J Appell')"
```

The last line runs the SQL statement we assigned to SQLQuery, adding the new record to the database.

```
dbConnection.Execute(SQLQuery)
```

Note If a user enters an apostrophe (like the one found in "can't", "won't", and "doesn't") into the *showform.asp* form, it will cause a syntax error when ASP tries to execute the SQL statement. The way around this is to add code to replace a single apostrophe with two apostrophes. For example you might want to change Request.Form("rpt_info") to Replace(Request.Form("rpt_info"),"'","''"). See your ASP documentation for more information.

Now let's look at the contents of the <SCRIPT> element:

```
alert("Point added successfully! Reload the map to see your changes.");
window.close();
```

The first line displays an alert telling users to reload the map to see their changes. The second line closes the form window, leaving only the original map window.

Creating an HTML Page to Display the Map

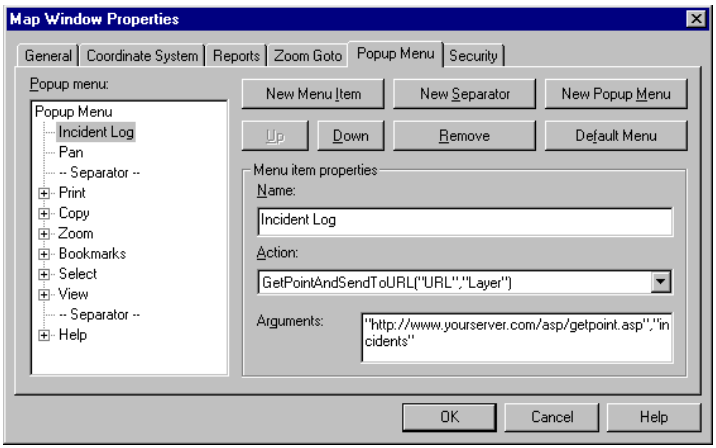
The last step is to create an HTML page to display our map. We'll use the *parcel_map.htm* file from the previous example, modifying the <H1> and the short paragraph of descriptive text, but leaving the rest of the file unchanged:

```
<HTML>
<HEAD>
   <TITLE>ASP Example</TITLE>
</HEAD>
<BODY>
<!-- Only the next two lines are different -->
<H1>Modifying a Database via the Map</H1>
<P>This example uses <b>ASP</b> to update a database map
resource</P>
<!-- embedded map -->
<OBJECT ID="myMap" WIDTH=600 HEIGHT=250
     CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
   <PARAM NAME="URL" VALUE="http://www.yourserver.com/maps/Starter-
       App.mwf?ReportTarget=Right>
   <EMBED SRC="http://www.yourserver.com/maps/StarterApp.mwf?Re-
       portTarget=RightNAME="myMap" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Creating a Custom Menu Item

As we designed it, our report has a few problems. The first is that it requires too many mouse clicks: users have to select View ➤ Reports from the map-window popup menu, then they have to select Incidents (ASP) from the list, then they have to clear the alert box that tells them to select a point, then they have to digitize the point. The other problem is that because our report isn't associated with a layer, users can add items to the Incidents map layer even when that layer isn't visible.

We can solve both of these problems by creating a custom menu item that takes the place of the report. We do so by selecting the following options on the Popup Menu tab of the Map Window Properties dialog box:



Dialog box specifications for 'Incident Log' menu item

Here are descriptions of how we used the options on the Popup Menu tab:

New Menu Item	Creates a new popup menu item below the item selected in the Popup Menu list.
Name	Specifies the name of the menu item as it will appear in the Viewer. Our menu item is named "Incident Log."
Action	Specifies the task to be performed by the menu item. We selected "GetPointAndSendToURL" from the drop-down list.
Arguments	Specifies arguments to use with the selected action, in this case the path to *getpoint.asp* and the name of the layer we want to add data to.

When users select Incident Log from the popup menu, they will immediately be able to enter a point, thus bypassing several mouse clicks. Also, if the Incidents layer is not visible because the map is zoomed outside of the layer's display range, the Incidents Log menu item will be unavailable.

Accessing Your Application with the Viewer API

For more info...
Refer to the *Autodesk MapGuide Viewer API Help*.

Because the Incident Log application runs in a separate instance of the browser, it does not have programmatic access to the map window. This means the application cannot refresh the map automatically. (That's why we have a JavaScript alert() box telling the user to reload the map manually.)

To solve this problem and to avoid having the user need to reload the map manually, use the Viewer API to access the Incident Log application. Instead of creating a report or custom menu item, add a button or other interface element to the HTML page hosting the map (or to a frame or child window with programmatic access to that page). The button should invoke a JavaScript function that does the following:

- Uses the digitizePoint() method to get the coordinates of a user-specified point.
- Invokes *getpoint.asp*, passing the point coordinates as URL parameters.
- Refreshes the map after *getpoint.asp*, *showform.asp*, and *insert.asp* have completed their work.

Now that you have seen how to update your databases using report scripts, read the next chapter to learn how to update SDFs using the Autodesk MapGuide SDF Component Toolkit.

Using the SDF Component Toolkit to Modify Spatial Data

In This Chapter 5

- About the SDF Component Toolkit
- Working with SDF files
- Performing common tasks with the Toolkit
- Updating SDFs via the map
- Visual Basic Examples

You can use the Autodesk MapGuide SDF Component Toolkit to create server-side applications that read and modify existing SDF files. These applications can interact with client-side scripts, allowing for dynamic updates based on user input. For example, you could create an application that lets users add polygon lot lines or points of interest to a map from their browser. This chapter introduces the Toolkit and provides an example of updating SDFs via the map.

About the SDF Component Toolkit

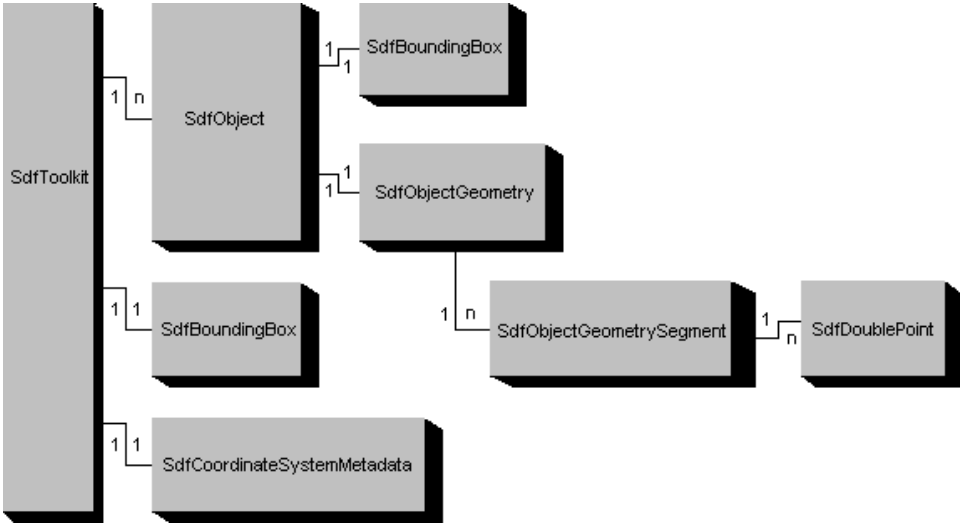
For more info...
Refer to the *Autodesk MapGuide SDF Component Toolkit Help* for a description of SDFs and their contents.

Autodesk MapGuide SDF Component Toolkit is a set of COM objects for reading and writing Spatial Data Files (SDF), Spatial Index Files (SIF), and Key Index Files (KIF). It provides the feature-by-feature control over SDF contents that Autodesk MapGuide SDF Loader does not provide. You can access SDF Component Toolkit objects in development environments such as C++, Visual Basic, VBA, VBScript, Java, JavaScript, Active Server Pages (ASP), ColdFusion, and the Common Gateway Interface (CGI).

You can install the SDF Component Toolkit from either the Autodesk MapGuide Author or Autodesk MapGuide Server CD. For detailed information about the SDF Component Toolkit, refer to the *Autodesk MapGuide SDF Component Toolkit Help*, an online help file that comes with the Toolkit.

Toolkit Objects

The Autodesk MapGuide SDF Component Toolkit contains the following objects:



The following sections describe each object. For a list of the methods and properties associated with each object, refer to the SDF Component Toolkit online help.

SdfToolkit

The SdfToolkit object represents an SDF file, which is identified by the SdfToolkit.Name property. All operations on the SDF file must be carried out after calling the SdfToolkit.Open method and before calling the SdfToolkit.Close method. The following information describes how to access and modify map features in the SDF.

Accessing map features

The SdfToolkit object contains SdfObject objects, which represent map features in the SDF file. There are three ways to access these map features.

- *Sequential search* After calling BeginSequentialSearch, call SearchToNextObject to iterate to the next map feature in sequence. Call EndSearch to conclude the search.
- *Spatially indexed search* After calling BeginSpatialIndexSearch, call SearchToNextObject to iterate to the next map feature that matches type and location criteria. Call EndSearch to conclude the search. This technique uses the SIF file, which must always co-exist with the SDF.
- *Key indexed search* After calling BeginKeyIndexSearch, call SearchToNextObject to iterate to the next map feature that matches a given key. Call EndSearch to conclude the search. Note that you cannot perform a key-indexed search unless the SDF file has an associated KIF file.

Modifying map features

To update an existing map feature in the SDF, you delete it from the SDF file and replace it with a new one. To do this, you can get a copy of the feature from the Toolkit, use SdfToolkit.DeleteObject to delete the feature from the file, modify the geometry in the copy of the feature, then call SdfToolkit.AddObject to add it back to the SDF file. Note that you can add and delete map features during an update process only, which is initiated by a call to SdfToolkit.BeginUpdate and concluded by a call to SdfToolkit.EndUpdate.

SdfObject

The SdfObject object represents a map feature in an SDF file. It contains an SdfObjectGeometry object, which represents the map feature's geometry data.

SdfObjectGeometry

The SdfObjectGeometry object stores the geometry data of a map feature in an SDF file. It is an indexed collection of SdfObjectGeometrySegment objects. The index of the first SdfObjectGeometrySegment object is 0.

SdfObjectGeometrySegment

The SdfObjectGeometrySegment object represents a segment of geometry data for an SdfObject object. It is an indexed collection of SdfDoublePoint objects. The index of the first SdfDoublePoint object is 0. SdfObjectGeometrySegment objects are collected in SdfObjectGeometry objects.

SdfDoublePoint

The SdfDoublePoint object represents a point in an SDF file. SdfDoublePoint objects are collected in SdfObjectGeometrySegment objects.

SdfBoundingBox

The SdfBoundingBox object represents a bounding box, which is the smallest rectangle that can be drawn around an individual map feature (including all of its constituent elements), an SDF file (including all of its map features), or the limits of a spatially indexed search.

SdfCoordinateSystemMetadata

The SdfCoordinateSystemMetadata object stores all elements of the coordinate system metadata of an SDF file, including the coordinate system code.

Status Codes

As with all applications, you should verify the status code returned from the current method before calling the next method. This is good programming practice that can reduce testing and debugging time.

When an SDF Component Toolkit method executes, it reports its status by returning an HRESULT code if you are coding in C++. If you are coding in Visual Basic, you call Err.Number to retrieve the status code, where "Err" is the name of the standard VB error object. In JavaScript and ColdFusion, the system automatically captures the status code and displays an error message.

Note In some environments, such as JavaScript, VBScript, or ColdFusion, you must refer to status codes by number. In other environments, such as Visual Basic, you can use their symbolic names.

Status codes are organized into the following three groups.

■ Spatial Data File status codes	sdf*xx*	0x80041*nnn*
■ Spatial Index File status codes	sif*xx*	0x80042*nnn*
■ Key Index File status codes	kif*xx*	0x80043*nnn*

For a list of the status codes in each of these groups, refer to the *Autodesk MapGuide SDF Component Toolkit* help.

Enumerated Constants

An enumerated constant group is used where a limited set of arguments is allowed for a parameter.

Note In some environments, such as JavaScript, VBScript, or ColdFusion, you must refer to enumerated constants by number. In other environments, such as Visual Basic, you can use their symbolic names.

There are four types of enumerated constants:

- SdfFileType constants
- SdfOpenFlags constants
- SdfObjectClass constants
- SdfObjectType constants

For a list of the enumerated constants and their values, refer to the *Autodesk MapGuide SDF Component Toolkit Help*.

Working with SDF Files

This section provides information about SDFs and the issues you need to be aware of when you work with them. For a complete discussion about the contents of SDF files, refer to the *Autodesk MapGuide SDF Component Toolkit Help*.

Indexing

SDF data is indexed in associated Spatial Index Files (SIF) and Key Index Files (KIF). Whenever the SDF Loader creates an SDF file, it creates a corresponding SIF file. Optionally, it creates a KIF file as well. SIF and KIF files have the same file names as their associated SDF files. SIF and KIF files are used by Autodesk MapGuide SDF Component Toolkit to access map features.

To create a KIF file for an existing SDF file, use SDF Loader or SDF Component Toolkit to convert the existing SDF to a new SDF, and specify that a KIF file should be output as well. Note that the output SDF in this process has to be named differently from the source. When the process is finished, you can delete the source and rename the output SDF, SIF, and KIF files with the original name.

Editing

SDF is a binary file format. To edit SDF files directly, you need Autodesk MapGuide SDF Component Toolkit, but you can edit them indirectly with Autodesk MapGuide SDF Loader. First, use SDF Loader to convert the data to SDL, which you can edit with a text editor, and then use SDF Loader again to convert the result to SDF. You can also edit the file that the SDF data originally came from. First modify it in its native application, and then use SDF Loader to convert the result to SDF.

SDF Pitfalls

The following list explains some pitfalls you might encounter when working with SDFs.

Fragmented Data

Deleting many map features from an SDF file can fragment its data. To compact a fragmented SDF file, use SDF Loader or the Toolkit to convert the fragmented SDF file to a new SDF file with a different name. You can then delete the old SDF and rename the new SDF with the original name.

Read and Write Issues

You cannot open an SDF file for writing while Autodesk MapGuide Server is accessing it.

You cannot open an SDF file even for reading if it is open for writing by another application, and neither can Autodesk MapGuide Server. If Server tries to access an SDF to service a request while the file is open for writing, the request fails, and Server returns the following error message.

Unable to open Spatial Data File…

To avoid such conflicts, you should modify an online SDF by editing a copy of it. When you are finished, replace the original with the modified copy.

Mismatched Precision

If the floating point precision for a given SDF and its corresponding SIF do not agree, the Toolkit returns the status code sdfPrecisionMismatch when you open the SDF file for writing. To correct this condition, you can generate a new SDF file from the existing one using either Autodesk MapGuide SDF Loader with the /COORDPREC switch set or SDF Component Toolkit with the SdfToolkit.Precision property set. The possible values are 32 or 64. Setting this

value for a new SDF file sets it for the corresponding SIF file automatically.

Note that precision mismatch is a possible problem only with SDF files created with Autodesk MapGuide SDF Loader 2.0 and earlier.

Unclosed Polygons

While writing to an SDF file using the SdfObject.SetGeometry method, if the Type argument is sdfPolygonObject or sdfPolyPolygonObject, but the SdfObjectGeometry object you are writing is not a closed figure, the function reports sdfPolygonUnclosed. One solution is to close all unclosed polygons before you add them to the SDF file. Alternatively, you can append polygons to the SDF file with Autodesk MapGuide SDF Loader, which closes unclosed polygons as it adds them (if adding from a DWG or DXF file, it closes them only if the /PL2PG switch is set), or you can write an error handler in your Autodesk MapGuide SDF Component Toolkit application to close the offending polygon and try again.

Too Many Points

When reading or writing polylines and polygons, Autodesk MapGuide SDF Component Toolkit returns the status code sdfTooManyPoints if it detects a map feature with more points than the maximum allowed (6500 points for 16-bit applications; 16384 points for 32-bit applications). To repair this condition, use Autodesk MapGuide SDF Loader with the /GENERALIZE switch set to convert the original SDF file to a new one in which points per map feature are reduced to acceptable values.

Performing Common Tasks with the Toolkit

There are seven main tasks you will perform with the Toolkit:

- Creating an SDF file
- Retrieving map features
- Finding map features within a search area
- Finding map features by key
- Modifying map features
- Appending map features
- Modifying SDF files using collection objects

The SDF Component Toolkit online help provides examples of how to perform each of these tasks in Visual Basic. To get up and running quickly, you can copy the code from the examples in the help, and then paste it into your application and modify it.

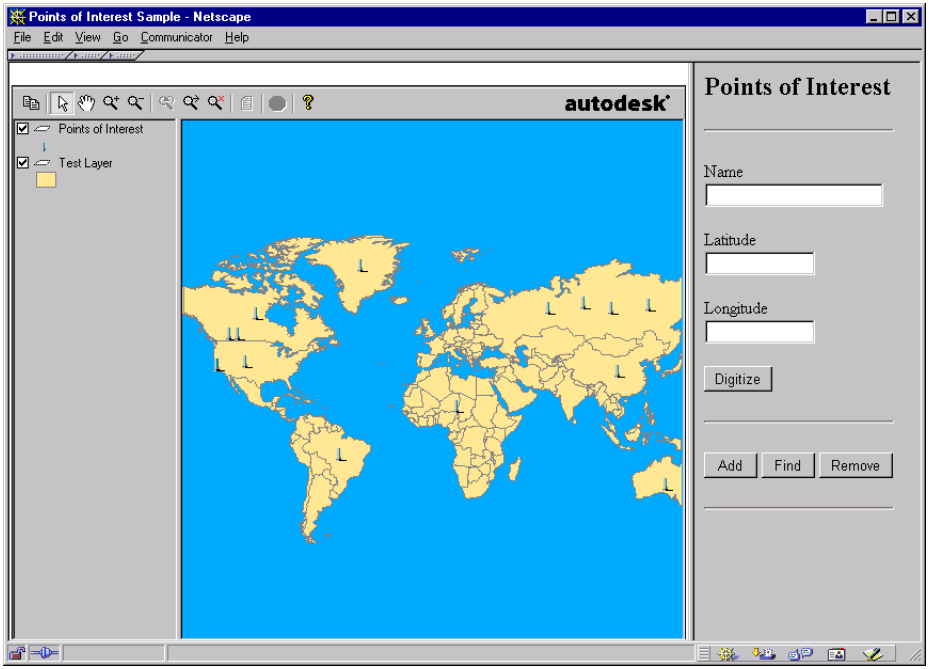
The next section describes an example application that takes you beyond these common tasks and shows you how to implement an advanced application with the Toolkit—updating SDFs dynamically via the map.

Updating SDF Files via the Map

In the previous chapter, we showed you ways to use reporting engines to add points to a database when the user clicks a point on the map. Similarly, you can use the SDF Component Toolkit to add points, polylines, or polygons to an SDF file when the user clicks the map. This section shows you how to create such an application.

Note that Autodesk MapGuide Release 5 contains a similar functionality called *redlining*. This means that users can add features to the map already (see "Custom Redlining Application" on page 49). However, the difference between the redlining functionality and the example in this section is that redlining changes are saved to the MWF on the user's computer, whereas the example in this section updates the data source for the map. With redlining, only the user sees the changes unless that MWF is then posted to the server. With the following SDF example, any map that uses the SDF as its data source will display the new points.

The example you are about to see is called Points of Interest. It uses Active Server Pages (ASP) and SDF Component Toolkit commands to allow users to add points of interest to the map.



Notice that the map is on the left, and there are controls on the right. To add a point, the user types a name for the point, clicks the Digitize button, clicks a point on the map, and then clicks the Add button. The new point and its name are added to *poi.sdf*, which is the SDF file on which the Points of Interest map layer is based. Points are represented on the map as L-shaped symbols.

To find the point again, you can type the name of the point and click the Find button. The map zooms to the point. To select the point, you can either click it, or you can right-click to display the popup menu, choose Select ➤ Select Map Features, and then select the name of the point from the list (if the name does not appear in the list, zoom out or click the Zoom Extents button to zoom all the way out, causing the point to be within the map view). To remove the point from the map, you type the name of a point and click the Remove button. The point is then deleted from *poi.sdf*.

The main page is called *maps_poi.htm*. It contains the following code, which sets up the frames.

```
<HTML>
<HEAD>
   <TITLE>Points of Interest Sample</TITLE>
</HEAD>
```

```
<FRAMESET COLS="75%,*" FRAMESPACING=0>
<FRAME SRC="map.htm" NAME="mapFrame" SCROLLING=NO MARGINHEIGHT=0
   MARGINWIDTH=0>
<FRAME SRC="poi.asp" NAME="poiFrame">
</FRAMESET>
</HTML>
```

Notice that the frame on the left, which contains the map, is called "mapFrame" and uses the page *map.htm*. The frame on the right, which contains the controls, is called "poiFrame" and uses *poi.asp*. First, take a look at the code for *map.htm*. This file will perform the following tasks:

- Embed the map.
- Set up the event observers that pass information from the onDigitizedPoint event to the appropriate function. You do this by defining a VBScript function for use by Internet Explorer and embedding the *MapGuideObserver5.class* file for use by Netscape.
- Define a function that takes the point from the observers and updates the text boxes with the point's coordinates.

Map.htm

```
<HTML>
<HEAD>
  <TITLE>Points of Interest Map</TITLE>
</HEAD>
<BODY>

<SCRIPT LANGUAGE="VBScript">
// Send onDigitizedPoint events from the ActiveX Control to the
// event-handling function
Sub map_onDigitizedPoint(Map, Point)
  onDigitizedPoint Map, Point
End Sub
</SCRIPT>

<SCRIPT LANGUAGE="JavaScript">
// Determine browser...
bName = navigator.appName;
bVer = parseInt(navigator.appVersion);

if (bName == "Netscape" && bVer >= 4)
  ver = "n4";
else if (bName == "Microsoft Internet Explorer" && bVer >= 4)
  ver = "e4";
else ver = "other";

// ...if Netscape, embed event observer
if (ver == "n4")
{
```

Map.htm (*continued*)

```
  document.write("<APPLET CODE=\"MapGuideObserver5.class\"
    WIDTH=2 HEIGHT=2 NAME=\"obs\" MAYSCRIPT>");
  document.write("</APPLET>");
}

// Now create the event-handling function. The function updates the
// lat & lon text boxes with the coordinates of the point the user
// clicks. Note that the name of this function must be the same as
// the event name so that the MapGuideObserver5 observer can find it.
function onDigitizedPoint(map, point)
{
  if (ver == "n4")
  {
    parent.poiFrame.document.pointForm.pointLat.value = point.getX();
    parent.poiFrame.document.pointForm.pointLon.value =
point.getY();
  }
  else
  {
    parent.poiFrame.pointForm.pointLat.value = point.getX();
    parent.poiFrame.pointForm.pointLon.value = point.getY();
  }
}
</SCRIPT>

//Embed the map with both the OBJECT tag and the EMBED tag so that
//it can be used by both browsers
<OBJECT ID="map" WIDTH="100%" HEIGHT="100%"
  CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
  <PARAM NAME="URL" VALUE="http://yourserver.com/maps/poi.mwf">
  <PARAM NAME="Lat" VALUE="0">
  <PARAM NAME="Lon" VALUE="0">
  <PARAM NAME="MapScale" VALUE="0">
  <PARAM NAME="MapWidth" VALUE="0">
  <PARAM NAME="Units" VALUE="M">
  <PARAM NAME="ToolBar" VALUE="On">
  <PARAM NAME="StatusBar" VALUE="On">
  <PARAM NAME="LayersViewWidth" VALUE="150">
  <PARAM NAME="DefaultTarget" VALUE="">
  <PARAM NAME="ErrorTarget" VALUE="">
  <PARAM NAME="ObjectLinkTarget" VALUE="">
  <PARAM NAME="ReportTarget" VALUE="">
  <PARAM NAME="URLList" VALUE="Off">
  <PARAM NAME="URLListTarget" VALUE="">
  <PARAM NAME="AutoLinkLayers" VALUE="">
  <PARAM NAME="AutoLinkTarget" VALUE="">
  <PARAM NAME="AutoLinkDelay" VALUE="20">
  <!-- in actual source, EMBED tag is on a single line -->
  <EMBED SRC="http://yourserver.com/maps/poi.mwf?URL=
```

For more info...
Refer to the *Viewer API Help* for information about the parameters you can use to control the display of the map when you embed it.

Map.htm (*continued*)

```
    http://yourserver.com/maps/poi.mwf&Lat=0&Lon=0&MaScale=0
    &Width=0&Units=M&ToolBar=On&StatusBar=On&LayersViewWidth=150
    &DefaultTarget=&ErrorTarget=&ObjectLinkTarget=&ReportTarget=
    &URLList=Off&URLListTarget=&AutoLinkLayers=&AutoLinkTarget=
    &AutoLinkDelay=20" NAME="map" WIDTH="100%" HEIGHT="100%">
</OBJECT>
</BODY>
</HTML>
```

So far, nothing too complex. The map is now embedded, the event observers are set up, and the onDigitizedPoint function is in place to update the Lat and Lon boxes with the coordinates from the point the user clicks. Now, take a look at the ASP code in *poi.asp*, which drives the controls in the right-hand frame. This code sets up the controls and the functions behind them. This is where you will see the SDF Component Toolkit commands. Note the use of the ASP method server.CreateObject() throughout the code; for more information about ASP, see "Summary of ASP Objects, Components, and Events" on page 103.

Poi.ASP

```
<%@ LANGUAGE="JavaScript" %>
<HTML>
<HEAD>
  <TITLE>Points of Interest</TITLE>
</HEAD>

<BODY BGCOLOR="#C0C0C0">
<%
// First, set up the variables
var op = Request.Form("op");// Operation being performed, hidden
form field
var pointName = "";// Value of Name field on the form
var pointLat = "";// Value of the Lat field on the form
var pointLon = "";// Value of the Lon field on the form
var msgText = "";// Message text to display at bottom of page
var actionOp = "";// Action to perform after successful operation
var zoomToLat = 0.0;// Lat to zoom to, used for Find operation.
var zoomToLon = 0.0;// Lon to zoom to, used for Find operation.

// Now, use the SDF Component Toolkit commands to drive the controls.
// This code is all wrapped within an if statement that verifies
// whether a valid command (add, find, or remove) has been issued
// before it creates an instance of the SDF Component Toolkit.
if (op == "Add" || op == "Find" || op == "Remove")
{
  pointName = Request.Form("pointName");
```

Poi.ASP (*continued*)

```
if (pointName != "")
{
  // Create an instance of the SDF Component Toolkit
  var sdfToolKit = Server.CreateObject("Autodesk.MgSdfToolkit.1");

  // If the command is Add or Remove, open the SDF for read/write.
  // If the command is Find, open the SDF as read-only.
  if (op == "Add" || op == "Remove")
  {
    //Use the constants "32 | 2" to indicate sdfOpenUpdate and
    //sdfOpenExisting. These constants open for read/write and
    //report errors if the file doesn't exist.
    sdfToolKit.Open("c:\\sdf\\poi.sdf", 32 | 2, true);

    if (op == "Add") //add the point
    {
      pointLat = parseFloat(Request.Form("pointLat"));
      pointLon = parseFloat(Request.Form("pointLon"));

      // Set up the variables for building the point. A point
      // in the SDF Component Toolkit follows the object
      // hierarchy (in shorthand) of
      // object.geometry.segment.point.
      // Proceed only if the lat and lon values are valid.
      if (!isNaN(pointLat) && !isNaN(pointLon))
      {
        var sdfObject =
              Server.CreateObject("Autodesk.MgSdfObject.1");
        var sdfGeometry =
              Server.CreateObject("Autodesk.MgSdfObjectGeome-
                  try.1");
        var sdfSegment =
              Server.CreateObject("Autodesk.MgSdfObjectGeometry-
                   Segment.1");
        var sdfPoint =
          Server.CreateObject("Autodesk.MgSdfDoublePoint.1");

        // Now build the point into an SDF object. Use the text
        // in the Name field for both the name and the key;
        // leave the URL empty.
        sdfPoint.SetCoordinates(pointLat, pointLon);
        sdfSegment.Add(sdfPoint);
        sdfGeometry.Add(sdfSegment);
        sdfObject.SetGeometry(0, sdfGeometry);
        sdfObject.Name = pointName;
        sdfObject.Key = pointName;
        sdfObject.Url = "";

        // The object is built. Now add it to the SDF.
        sdfToolKit.BeginUpdate();
```

For more info...
In the *SDF Component Toolkit Help*, look up:
SdfOpenFlags constants

For more info...
In the *SDF Component Toolkit Help*, look up:
object model

Poi.ASP (*continued*)

```
            sdfToolKit.AddObject(sdfObject);
            sdfToolKit.EndUpdate();

            clearVars();
            actionOp = "UpdateMap";
            msgText = "Point added, updating map.";
          }
          else // Invalid lat/lon values
            msgText = "Lat and Lon floating point quantities must
                be specified."
        }
        else //if not the Add command, remove the point
        {
          var sdfObject = findObject(sdfToolKit, pointName);

          if (sdfObject != null)
          {
            sdfToolKit.BeginUpdate();
            sdfToolKit.DeleteObject(sdfObject);
            sdfToolKit.EndUpdate();

            clearVars();
            actionOp = "UpdateMap"; // Refreshes the map
            msgText = "Point removed, updating map.";
          }
          else
            msgText = "Point not found.";
        }
      }
      //If it's the Find command, open the SDF as read-only by setting
      //the second parameter to 1 (sdfOpenRead) instead of 2
      //(sdfOpenUpdate).
      else if (op == "Find")
      {
        var sdfObject = null;

        sdfToolKit.Open("c:\\sdf\\poi.sdf", 1, true);
        sdfObject = findObject(sdfToolKit, pointName);

        if (sdfObject != null)
        {
          var sdfPoint = sdfObject.Geometry.GetAt(0).GetAt(0);
          zoomToLon = sdfPoint.X;
          zoomToLat = sdfPoint.Y;
          clearVars();
          actionOp = "ZoomToPoint"; // Zooms to the point on the map
          msgText = "Zooming to point.";
        }
        else
```

Poi.ASP (*continued*)

```
          msgText = "Point not found.";
      }
      sdfToolKit.Close();
    }
    else
      msgText = "A name must be specified."
}
else if (Request.Count > 0)
  msgText = "Unrecognized command.";
%>

<%
function findObject(openSdf, objKey) // The actual search
{
  var sdfObject = null;

  openSdf.BeginKeyIndexSearch(objKey);
  sdfObject = openSdf.SearchToNextObject();
  openSdf.EndSearch();

  return sdfObject;
}

function clearVars()
{
  pointName = "";
  pointLat = "";
  pointLon = "";
  op = "";
}
%>

// This next section creates the content of the frame.
<H2>Points of Interest</H2>
<HR>
<FORM METHOD="POST" NAME="pointForm" TARGET="_self"
    ACTION="poi.asp">
  <INPUT TYPE="hidden" NAME="op" VALUE="None">
  <DIV ALIGN="left">
  <P>
  <LABEL FOR="fp1">Name<BR></LABEL>
  <INPUT TYPE="text" NAME="pointName" VALUE="<%=pointName%>"
    SIZE="20" maxlength="255" ID="fp1">
  </P>
  </DIV>
  <DIV ALIGN="left">
  <P>
  <LABEL FOR="fp2">Latitude<BR></LABEL>
  <INPUT TYPE="text" NAME="pointLat" VALUE="<%=pointLat%>"
```

Poi.ASP (*continued*)

```
    SIZE="12" ID="fp2">
  </P>
  </DIV>
  <DIV ALIGN="left">
  <P>
  <LABEL FOR="fp3">Longitude<BR></LABEL>
  <INPUT TYPE="text" NAME="pointLon" VALUE="<%=pointLon%>"
    SIZE="12" ID="fp3">
  </P>
  </DIV>
  <P>
  <INPUT TYPE="button" VALUE="Digitize" NAME="digitizePoint"
    LANGUAGE="JavaScript" ONCLICK="digitizeIt()">
  </P>
  <HR>
  <P>
  <INPUT TYPE="button" WIDTH="50" VALUE="Add" NAME="addPoint"
    WIDTH="50" LANGUAGE="JavaScript"
    ONCLICK="pointForm.op.value='Add'; pointForm.submit()">
  <INPUT TYPE="button" WIDTH="50" VALUE="Find" NAME="findPoint"
    LANGUAGE="JavaScript"
    ONCLICK="pointForm.op.value='Find';   pointForm.submit()">
  <INPUT TYPE="button" WIDTH="50" VALUE="Remove"
    NAME="removePoint"   LANGUAGE="JavaScript"
    ONCLICK="pointForm.op.value = 'Remove'; pointForm.submit()">
  </P>
</FORM>
<HR>
<%=msgText%>

<SCRIPT LANGUAGE="JavaScript">
// Determine browser...
bName = navigator.appName;
bVer = parseInt(navigator.appVersion);

if (bName == "Netscape" && bVer >= 4)
  ver = "n4";
else if (bName == "Microsoft Internet Explorer" && bVer >= 4)
  ver = "e4";
else
  ver = "other";

function getThisMap()
{
  if (ver == "n4")
    return parent.mapFrame.document.map;
  else
    return parent.mapFrame.map;
}
```

Poi.ASP (*continued*)

```
// Following is the function called by the Digitize button. If the
// browser is Navigator, it will send the onDigitizedPoint event to
// the MapGuideObserver5.class observer (the "obs" variable). If the
// browser is Internet Explorer, it will know to look for an observer
// method with the same name as the event, which we defined with
// VBScript in map.htm.
function digitizeIt()
{
  if (ver == "n4")
    getThisMap().digitizePoint(parent.mapFrame.document.obs);
  else
    getThisMap().digitizePoint();
}

function updateMap()      // Updates the map after an Add or Remove
{
  getThisMap().getMapLayer("POI").setRebuild(true);
  getThisMap().refresh();
}

function zoomToPoint()    // Zooms to the point after a Find
{
  getThisMap().zoomWidth(<%=zoomToLat%>, <%=zoomToLon%>, 1000, "Mi");
  getThisMap().refresh();
}
</SCRIPT>

<%
//When the resulting HMTL loads in the browser, these last few lines
//of ASP code call updateMap() or zoomToPoint(), depending on the
//operation performed when the ASP was executed.

if (actionOp == "UpdateMap")
{
  Response.Write("<SCRIPT LANGUAGE=\"JavaScript\">");
  Response.Write("updateMap();");
  Response.Write("</SCRIPT>");
}
else if (actionOp == "ZoomToPoint")
{
  Response.Write("<SCRIPT LANGUAGE=\"JavaScript\">");
  Response.Write("zoomToPoint();");
  Response.Write("</SCRIPT>");
}
%>

</BODY>
</HTML>
```

Once you understand the object hierarchy, using the SDF Component Toolkit is very straightforward. This example used ASP, but you can apply similar techniques using ColdFusion or another language. Please go to the customer sites pages (*www.autodesk.com/mapguidedemo*) to see more applications and add your own applications to share with others.

Visual Basic Examples

The rest of this chapter contains three applications written in Visual Basic. They illustrate the following tasks:

- Converting an SDF file
- Getting information about an SDF file
- Copying an SDF file

Converting To an SDF File

The ConvertSDF example shows how to implement a proprietary data converter. This code accesses a text file that contains necessary coordinate information and other attributes for SDF objects and creates an SDF file from it.

Converting To an SDF File

```
VERSION 5.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0";
"RICHTX32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0";
"comdlg32.ocx"
Begin VB.Form frmLab2
   Caption         =   "Form1"
   ClientHeight    =   3360
   ClientLeft      =   48
   ClientTop       =   276
   ClientWidth     =   5820
   LinkTopic       =   "Form1"
   ScaleHeight     =   3360
   ScaleWidth      =   5820
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton btnDelete
      Cancel          =   -1  'True
      Caption         =   "Delete Feature"
      Height          =   288
      Left            =   4440
      TabIndex        =   7
      Top             =   1320
      Width           =   1212
```

Converting To an SDF File (*continued*)

```
   End
   Begin VB.CommandButton btnConvert
      Caption         =   "Convert"
      Height          =   288
      Left            =   4440
      TabIndex        =   6
      Top             =   960
      Width           =   1212
   End
   Begin VB.CommandButton btnBrowseTxt
      Caption         =   "Browse"
      Height          =   288
      Left            =   4440
      TabIndex        =   5
      Top             =   120
      Width           =   1212
   End
   Begin VB.TextBox txtTxtName
      Height          =   288
      Left            =   120
      TabIndex        =   4
      Text            =   "d:\work\mapguide\water.txt"
      Top             =   120
      Width           =   4212
   End
   Begin VB.CommandButton btnExit
      Caption         =   "Exit"
      Height          =   288
      Left            =   4440
      TabIndex        =   3
      Top             =   3000
      Width           =   1212
   End
   Begin MSComDlg.CommonDialog cdOpen
      Left            =   4920
      Top             =   2520
      _ExtentX        =   677
      _ExtentY        =   677
      _Version        =   393216
      CancelError     =   -1  'True
      DialogTitle     =   "Open SDF File"
      Filter          =   "SDF Files (*.sdf) | *.sdf"
      FilterIndex     =   1
   End
   Begin VB.TextBox txtSdfName
      Height          =   288
      Left            =   120
      TabIndex        =   2
      Text            =   "d:\work\mapguide\water.sdf"
```

Converting To an SDF File (*continued*)

```
      Top             =   480
      Width           =   4212
   End
   Begin VB.CommandButton btnBrowseSdf
      Caption         =   "Browse"
      Height          =   288
      Left            =   4440
      TabIndex        =   1
      Top             =   480
      Width           =   1212
   End
   Begin RichTextLib.RichTextBox txtMsg
      Height          =   2292
      Left            =   120
      TabIndex        =   0
      Top             =   960
      Width           =   4212
      _ExtentX        =   7430
      _ExtentY        =   4043
      _Version        =   393217
      Enabled         =   -1  'True
      ReadOnly        =   -1  'True
      ScrollBars      =   2
      TextRTF         =   $"frmLab2.frx":0000
      BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
         Name            =   "Courier New"
         Size            =   7.8
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
   End
End
Attribute VB_Name = "frmLab2"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub btnBrowseSdf_Click()
On Error GoTo ErrHandler

    cdOpen.Filter = "SDF Files (*.SDF)| *.SDF"
    cdOpen.FilterIndex = 1
    cdOpen.DialogTitle = "Save SDF File"
    'Show the open dialog box
```

Converting To an SDF File (*continued*)

```
    cdOpen.ShowSave
    txtSdfName.Text = cdOpen.FileName

ErrHandler:
    'Cancel was selected
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnBrowseTxt_Click()
On Error GoTo ErrHandler

    cdOpen.Filter = "Text Files (*.TXT)| *.TXT"
    cdOpen.FilterIndex = 1
    cdOpen.DialogTitle = "Open SDF File"

    'Show the open dialog box
    cdOpen.ShowOpen
    txtTxtName.Text = cdOpen.FileName

ErrHandler:
    'Cancel was selected
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnConvert_Click()
    'Check if there is a file name in edit boxes
    If (txtSdfName.Text = "") Or (txtTxtName.Text = "") Then
        ShowMessage ("Select the TXT and SDF files first!")
        Exit Sub
    End If

    Dim oTlkt As New SdfToolkit
    Dim oObj As New SdfObject
    Dim oGeom As New SdfObjectGeometry
    Dim oSeg As New SdfObjectGeometrySegment
    Dim oPnt As New SdfDoublePoint
    Dim strMsg As String
    Dim strObjType As String, strKey As String, strName As String,
strUrl As String, strCnt As String
    Dim X As Double, Y As Double
    Dim i As Long, j As Long

On Error GoTo ErrHandler

    'Open the input text file
    Open txtTxtName.Text For Input As #1
    'Open the sdf file in read-only mode
```

Converting To an SDF File (*continued*)

```
    oTlkt.Open txtSdfName.Text, sdfOpenUpdate Or sdfCreateAlways,
      True
    'Indicate update process
    oTlkt.BeginUpdate
    j = 0
    'Read from txt file till eof
    Do While Not EOF(1)
        'Get the feature data
        Line Input #1, strObjType
        Line Input #1, strKey               'Key
        Line Input #1, strName              'Name
        Line Input #1, strUrl               'Url
        Line Input #1, strCnt               'Vertex count

        'Read the geometry
        For i = 1 To Val(strCnt)
            Input #1, X
            Input #1, Y
            'Prepare a point feature
            oPnt.SetCoordinates X, Y
            'Add this point into the segment
            oSeg.Add oPnt
        Next i
        'Now add this segment into the geometry
        oGeom.Add oSeg
        'Put this geometry into the feature
        If strObjType = "POLYGON" Then
            oObj.SetGeometry sdfPolygonObject, oGeom
        ElseIf strObjType = "POLYLINE" Then
            oObj.SetGeometry sdfPolylineObject, oGeom
        ElseIf strObjType = "POINT" Then
            oObj.SetGeometry sdfPointObject, oGeom
        Else
            ShowMessage "Unknown feature in input file."
        End If

        'Set the feature properties
        oObj.Key = Trim(strKey)
        oObj.Name = Trim(strName)
        oObj.Url = Trim(strUrl)

        'Add this feature to the SDF file
        oTlkt.AddObject oObj
        j = j + 1

        'Clear the geometry before next use
        oGeom.RemoveAll
        oSeg.RemoveAll
    Loop
```

Converting To an SDF File (*continued*)

```
    'Wind up
    oTlkt.EndUpdate
    oTlkt.Close
    Close #1

   ShowMessage txtTxtName.Text & " converted to " & txtSdfName.Text
    ShowMessage "Total features converted: " & j
    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0

End Sub

Private Sub btnDelete_Click()
 'Check if there is a file name in edit box
    If txtSdfName.Text = "" Then
        ShowMessage ("Select an SDF file first!")
        Exit Sub
    End If

    Dim Key As String

    'Get the key from
    Key = InputBox("Enter the key of the feature to be deleted: ",
      "Key Input", "Lake")
    If Key = "" Then
        Exit Sub
    End If


    Dim oTlkt As New SdfToolkit
    Dim oBox As SdfBoundingBox
    Dim oObj As SdfObject
    Dim strMsg As String
    Dim i As Long
    Dim objFound As Boolean


On Error GoTo ErrHandler

    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenUpdate Or sdfOpenExisting,
True
    'Begin spatial search for polylines
    oTlkt.BeginKeyIndexSearch (Key)
```

Converting To an SDF File (*continued*)

```
    'Get first feature having key
    Set oObj = oTlkt.SearchToNextObject()
    'End search
    oTlkt.EndSearch

    objFound = Not (oObj Is Nothing)
    If Not (oObj Is Nothing) Then
        ShowMessage "Following feature is deleted"
        ShowMessage "Feature: " & i & " " & GetObjectTypeS-
          tring(oObj.Type)
        ShowMessage "   Key : " & oObj.Key
        ShowMessage "   Name: " & oObj.Name
        ShowMessage "   Url : " & oObj.Url
        'Delete this feature
        oTlkt.BeginUpdate
        oTlkt.DeleteObject oObj
        oTlkt.EndUpdate
    End If


    'If we come here, feature with specified key was not found
    If Not objFound Then
        ShowMessage ("Feature with specified key not found.")
    Else
        ShowMessage ("Feature with key " & Key & " deleted.")
    End If

    oTlkt.Close

    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub


Private Sub btnExit_Click()
    End
End Sub

Sub ShowMessage(Msg As String)

    txtMsg.Text = txtMsg.Text & Msg & vbCrLf

End Sub
```

Converting To an SDF File (*continued*)

```
Function GetObjectTypeString(ObjType As SdfObjectType) As String
    Select Case ObjType
        Case sdfPointObject:
            GetObjectTypeString = "POINT"
        Case sdfPolygonObject:
            GetObjectTypeString = "POLYGON"
        Case sdfPolylineObject:
            GetObjectTypeString = "POLYLINE"
        Case sdfPolyPolylineObject:
            GetObjectTypeString = "POLYPOLYLINE"
        Case sdfPolyPolygonObject:
            GetObjectTypeString = "POLYPOLYGON"
    End Select

End Function
```

Getting Information about an SDF File

The SDFInfo example code shows how to open and access an SDF file to retrieve its information, such as precision, key length, bounding box, etc. It also shows how to search for features within the SDF file using sequential search, spatial search, and key-indexed search.

Getting Information About an SDF File

```
VERSION 5.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0";
"RICHTX32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0";
"comdlg32.ocx"
Begin VB.Form frmLab1
   Caption         =   "Form1"
   ClientHeight    =   3360
   ClientLeft      =   48
   ClientTop       =   276
   ClientWidth     =   5820
   LinkTopic       =   "Form1"
   ScaleHeight     =   3360
   ScaleWidth      =   5820
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton btnExit
      Caption         =   "Exit"
      Height          =   288
      Left            =   4440
      TabIndex        =   7
      Top             =   3000
      Width           =   1212
   End
```

Getting Information About an SDF File (*continued*)

```
   Begin VB.CommandButton btnSrchKey
      Caption         =   "Key Search"
      Height          =   288
      Left            =   4440
      TabIndex        =   6
      Top             =   1800
      Width           =   1212
   End
   Begin VB.CommandButton btnSrchSpat
      Caption         =   "Spatial Search"
      Height          =   288
      Left            =   4440
      TabIndex        =   5
      Top             =   1440
      Width           =   1212
   End
   Begin VB.CommandButton btnSrchSeq
      Caption         =   "Seq Search"
      Height          =   288
      Left            =   4440
      TabIndex        =   4
      Top             =   1080
      Width           =   1212
   End
   Begin VB.CommandButton btnShowInfo
      Caption         =   "Show Info"
      Height          =   288
      Left            =   4440
      TabIndex        =   3
      Top             =   720
      Width           =   1212
   End
   Begin MSComDlg.CommonDialog cdOpen
      Left            =   4920
      Top             =   2280
      _ExtentX        =   677
      _ExtentY        =   677
      _Version        =   393216
      CancelError     =   -1  'True
      DialogTitle     =   "Open SDF File"
      Filter          =   "SDF Files (*.sdf) | *.sdf"
      FilterIndex     =   1
   End
   Begin VB.TextBox txtSdfName
      Height          =   288
      Left            =   120
      TabIndex        =   2
      Text            =   "d:\work\mapguide\redline.sdf"
      Top             =   120
```

Getting Information About an SDF File (*continued*)

```
      Width           =   4212
   End
   Begin VB.CommandButton btnBrowse
      Caption         =   "Browse"
      Height          =   288
      Left            =   4440
      TabIndex        =   1
      Top             =   120
      Width           =   1212
   End
   Begin RichTextLib.RichTextBox txtMsg
      Height          =   2652
      Left            =   120
      TabIndex        =   0
      Top             =   600
      Width           =   4212
      _ExtentX        =   7430
      _ExtentY        =   4678
      _Version        =   393217
      Enabled         =   -1  'True
      ReadOnly        =   -1  'True
      ScrollBars      =   2
      TextRTF         =   $"frmLab1.frx":0000
      BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
         Name            =   "Courier New"
         Size            =   7.8
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
   End
End
Attribute VB_Name = "frmLab1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub btnBrowse_Click()
On Error GoTo ErrHandler

    'Show the open dialog box
    cdOpen.ShowOpen
    txtSdfName.Text = cdOpen.FileName
ErrHandler:
    'Cancel was selected
```

Getting Information About an SDF File (*continued*)

```
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnExit_Click()
    End
End Sub

Private Sub btnShowInfo_Click()

    'Check if there is a file name in edit box
    If txtSdfName.Text = "" Then
        ShowMessage ("Select an SDF file first!")
        Exit Sub
    End If


    Dim oTlkt As New SdfToolkit
    Dim oBox  As SdfBoundingBox
    Dim strMsg As String

On Error GoTo ErrHandler

    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenRead, True

    'Get SDF name
    strMsg = oTlkt.Name
    ShowMessage "SDF File opened: " & strMsg

    'Get description
    strMsg = oTlkt.Description
    ShowMessage "Description: " & strMsg

    'Get precision
    strMsg = oTlkt.Precision
    ShowMessage "Precision: " & strMsg & " bit"

    'Get key length
    strMsg = oTlkt.MaxKeyLength
    ShowMessage "Max Key length: " & strMsg

    'Get version
    strMsg = oTlkt.Version
    ShowMessage "Version: " & strMsg

    'Get extents
    strMsg = "Min LAT: " & oTlkt.BoundingBox.minY & vbCrLf & _
             "Min LON: " & oTlkt.BoundingBox.minX & vbCrLf & _
```

Getting Information About an SDF File (*continued*)

```
            "Max LAT: " & oTlkt.BoundingBox.maxY & vbCrLf & _
            "Max LON: " & oTlkt.BoundingBox.maxX

    ShowMessage "SDF Extents: " & vbCrLf & strMsg

    'Get total count of features
    strMsg = oTlkt.TotalObjects
    ShowMessage "Total features: " & strMsg

    'Check for feature classes present in this sdf
    ShowMessage "Contains Points: " & oTlkt.ContainsObjectClass(sd-
      fPointClass)
    ShowMessage "Contains Polylines: " & oTlkt.ContainsObject-
      Class(sdfPolylineClass)
    ShowMessage "Contains Polygons: " & oTlkt.ContainsObject-
      Class(sdfPolygonClass)

    'Close the toolkit
    oTlkt.Close

    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub

Private Sub btnSrchKey_Click()
'Check if there is a file name in edit box
    If txtSdfName.Text = "" Then
        ShowMessage ("Select an SDF file first!")
        Exit Sub
    End If

    Dim Key As String

    'Get the key from
    Key = InputBox("Enter the key of the feature: ", "Key Input",
"RedLine[144.111.8.96]")
    If Key = "" Then
        Exit Sub
    End If


    Dim oTlkt As New SdfToolkit
    Dim oBox As SdfBoundingBox
    Dim oObj As SdfObject
```

Getting Information About an SDF File (*continued*)

```
    Dim strMsg As String
    Dim i As Long
    Dim objFound As Boolean


On Error GoTo ErrHandler

    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenRead, True

    'Begin spatial search for polylines
    oTlkt.BeginKeyIndexSearch (Key)

    'Get first feature
    Set oObj = oTlkt.SearchToNextObject()
    objFound = Not (oObj Is Nothing)
    i = 1

    Do While Not (oObj Is Nothing)
        ShowMessage "Feature: " & i & " " & GetObjectTypeS-
tring(oObj.Type)
        ShowMessage "   Key : " & oObj.Key
        ShowMessage "   Name: " & oObj.Name
        ShowMessage "   Url : " & oObj.Url
        Set oObj = oTlkt.SearchToNextObject()
        i = i + 1
        DoEvents
    Loop

    'If we come here, feature with specified key was not found
    If Not objFound Then
        ShowMessage ("Feature with specified key not found.")
    Else
       ShowMessage (i - 1 & " features with key " & Key & " found.")
    End If

    'Close the toolkit
    oTlkt.EndSearch
    oTlkt.Close

    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub
```

Getting Information About an SDF File (*continued*)

```
Private Sub btnSrchSeq_Click()
'Check if there is a file name in edit box
    If txtSdfName.Text = "" Then
        ShowMessage ("Select an SDF file first.")
        Exit Sub
    End If


    Dim oTlkt As New SdfToolkit
    Dim oObj As SdfObject
    Dim strMsg As String
    Dim i As Long

On Error GoTo ErrHandler

    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenRead, True

    'Begin sequential search
    oTlkt.BeginSequentialSearch

    'Get first feature
    Set oObj = oTlkt.SearchToNextObject()

    i = 1
    While Not (oObj Is Nothing)
        ShowMessage "Feature: " & i & " " & GetObjectTypeS-
tring(oObj.Type)
        ShowMessage "   Key: " & oObj.Key
        ShowMessage "   Name: " & oObj.Name
        ShowMessage "   Url: " & oObj.Url
        Set oObj = oTlkt.SearchToNextObject()
        i = i + 1
        DoEvents
    Wend

    'Close the toolkit
    oTlkt.EndSearch
    oTlkt.Close

    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub
```

Getting Information About an SDF File (*continued*)

```
Private Sub btnSrchSpat_Click()
'Check if there is a file name in edit box
    If txtSdfName.Text = "" Then
        ShowMessage ("Select an SDF file first!")
        Exit Sub
    End If


    Dim oTlkt As New SdfToolkit
    Dim oBox As SdfBoundingBox
    Dim oObj As SdfObject
    Dim strMsg As String
    Dim i As Long


On Error GoTo ErrHandler

    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenRead, True

    'Get the SDF extents
    Set oBox = oTlkt.BoundingBox

    'Begin spatial search for polylines
    oTlkt.BeginSpatialIndexSearch sdfPolylineClass, oBox

    'Get first feature
    Set oObj = oTlkt.SearchToNextObject()

    i = 1

    While Not (oObj Is Nothing)
        ShowMessage "Feature: " & i & " " & GetObjectTypeS-
tring(oObj.Type)
        ShowMessage "   Key: " & oObj.Key
        ShowMessage "   Name: " & oObj.Name
        ShowMessage "   Url: " & oObj.Url
        Set oObj = oTlkt.SearchToNextObject()
        i = i + 1
        DoEvents
    Wend

    'Close the toolkit
    oTlkt.EndSearch
    oTlkt.Close

    Exit Sub
ErrHandler:
    'Display the error number/message
```

Getting Information About an SDF File (*continued*)

```
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub


Sub ShowMessage(Msg As String)

    txtMsg.Text = txtMsg.Text & Msg & vbCrLf

End Sub


Function GetObjectTypeString(ObjType As SdfObjectType) As String
    Select Case ObjType
        Case sdfPointObject:
            GetObjectTypeString = "POINT"
        Case sdfPolygonObject:
            GetObjectTypeString = "POLYGON"
        Case sdfPolylineObject:
            GetObjectTypeString = "POLYLINE"
        Case sdfPolyPolylineObject:
            GetObjectTypeString = "POLYPOLYLINE"
        Case sdfPolyPolygonObject:
            GetObjectTypeString = "POLYPOLYGON"
    End Select

End Function
```

Copying an SDF File

The CopySDF example shows how to open an existing SDF file and write its features to a new SDF file.

Copying an SDF File

```
VERSION 5.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0";
"RICHTX32.OCX"
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0";
"comdlg32.ocx"
Begin VB.Form frmLab2
   Caption         =   "Form1"
   ClientHeight    =   3360
   ClientLeft      =   48
   ClientTop       =   276
   ClientWidth     =   5820
   LinkTopic       =   "Form1"
```

Copying an SDF File (*continued*)

```
   ScaleHeight     =   3360
   ScaleWidth      =   5820
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton btnCopy
      Caption         =   "Copy"
      Height          =   288
      Left            =   4440
      TabIndex        =   6
      Top             =   960
      Width           =   1212
   End
   Begin VB.CommandButton btnBrowseInSdf
      Caption         =   "Browse"
      Height          =   288
      Left            =   4440
      TabIndex        =   5
      Top             =   120
      Width           =   1212
   End
   Begin VB.TextBox sdfInName
      Height          =   288
      Left            =   120
      TabIndex        =   4
      Text            =   "d:\work\mapguide\redline.sdf"
      Top             =   120
      Width           =   4212
   End
   Begin VB.CommandButton btnExit
      Caption         =   "Exit"
      Height          =   288
      Left            =   4440
      TabIndex        =   3
      Top             =   3000
      Width           =   1212
   End
   Begin MSComDlg.CommonDialog cdOpen
      Left            =   4920
      Top             =   2520
      _ExtentX        =   677
      _ExtentY        =   677
      _Version        =   393216
      CancelError     =   -1  'True
      DialogTitle     =   "Open SDF File"
      Filter          =   "SDF Files (*.sdf) | *.sdf"
      FilterIndex     =   1
   End
   Begin VB.TextBox sdfOutName
      Height          =   288
      Left            =   120
```

Copying an SDF File (*continued*)

```
      TabIndex        =   2
      Text            =   "d:\work\mapguide\redline1.sdf"
      Top             =   480
      Width           =   4212
   End
   Begin VB.CommandButton btnBrowseOutSdf
      Caption         =   "Browse"
      Height          =   288
      Left            =   4440
      TabIndex        =   1
      Top             =   480
      Width           =   1212
   End
   Begin RichTextLib.RichTextBox txtMsg
      Height          =   2292
      Left            =   120
      TabIndex        =   0
      Top             =   960
      Width           =   4212
      _ExtentX        =   7430
      _ExtentY        =   4043
      _Version        =   393217
      Enabled         =   -1  'True
      ReadOnly        =   -1  'True
      ScrollBars      =   2
      TextRTF         =   $"frmLab3.frx":0000
      BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
         Name            =   "Courier New"
         Size            =   7.8
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
   End
End
Attribute VB_Name = "frmLab2"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub btnBrowseInSdf_Click()
On Error GoTo ErrHandler

    cdOpen.Filter = "Text Files (*.SDF)| *.SDF"
    cdOpen.FilterIndex = 1
```

Copying an SDF File (*continued*)

```
    cdOpen.DialogTitle = "Open SDF File"

    'Show the open dialog box
    cdOpen.ShowOpen
    sdfInName.Text = cdOpen.FileName

ErrHandler:
    'Cancel was selected
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnBrowseOutSdf_Click()
On Error GoTo ErrHandler

    cdOpen.Filter = "SDF Files (*.SDF)| *.SDF"
    cdOpen.FilterIndex = 1
    cdOpen.DialogTitle = "Save SDF File"

    'Show the open dialog box
    cdOpen.ShowSave
    sdfOutName.Text = cdOpen.FileName

ErrHandler:
    'Cancel was selected
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnCopy_Click()
Dim msg As String

    'Check for filenames
    If (sdfInName.Text = "") Or (sdfOutName.Text = "") Then
        ShowMessage "You must select filenames first.'"
        Exit Sub
    End If

    Dim oTlktIn As New SdfToolkit
    Dim oTlktOut As New SdfToolkit
    Dim oObj As SdfObject
    Dim oBox As New SdfBoundingBox
   Dim xMin As Double, yMin As Double, xMax As Double, yMax As Double
    Dim i As Long

On Error GoTo ErrHandler

    'Open the input sdf file in readonly mode
    oTlktIn.Open sdfInName, sdfOpenRead, True
```

Copying an SDF File (*continued*)

```
    'Open the output sdf file for write/append mode
    oTlktOut.Open sdfOutName, sdfOpenUpdate Or sdfOpenAlways, True

    'Get insdf's extents
    Set oBox = oTlktIn.BoundingBox
    Debug.Print oBox.MinX, oBox.MinY, oBox.MaxX, oBox.MaxY

    'Set up search area that is 1/3 of input sdf
    xMin = oBox.MinX + Abs((Abs(oBox.MaxX) - Abs(oBox.MinX)) / 3)
    xMax = oBox.MaxX - Abs((Abs(oBox.MaxX) - Abs(oBox.MinX)) / 3)
    yMin = oBox.MinY + Abs((Abs(oBox.MaxY) - Abs(oBox.MinY)) / 3)
    yMax = oBox.MaxY - Abs((Abs(oBox.MaxY) - Abs(oBox.MinY)) / 3)
    oBox.SetExtent xMin, yMin, xMax, yMax

    'Start searching from insdf and writing to outsdf

    oTlktIn.BeginSpatialIndexSearch sdfAllObjectClasses, oBox
    oTlktOut.BeginUpdate

    Set oObj = oTlktIn.SearchToNextObject

    i = 0
    Do While Not (oObj Is Nothing)
        oTlktOut.AddObject oObj
        i = i + 1
        Set oObj = oTlktIn.SearchToNextObject
    Loop


    'Wind up
    oTlktIn.EndSearch
    oTlktOut.EndUpdate
    oTlktIn.Close
    oTlktOut.Close

    ShowMessage sdfInName.Text & " copied to " & sdfOutName.Text
    ShowMessage "Total features written: " & i


    Exit Sub

ErrHandler:
    'Display the error number/message
    MsgBox Err.Number & " : " & Err.Description
    'Reset the handler before exiting
    On Error GoTo 0
End Sub
```

Copying an SDF File (*continued*)

```
Private Sub btnExit_Click()
    End
End Sub

Sub ShowMessage(msg As String)

    txtMsg.Text = txtMsg.Text & msg & vbCrLf

End Sub


Function GetObjectTypeString(ObjType As SdfObjectType) As String
    Select Case ObjType
        Case sdfPointObject:
            GetObjectTypeString = "POINT"
        Case sdfPolygonObject:
            GetObjectTypeString = "POLYGON"
        Case sdfPolylineObject:
            GetObjectTypeString = "POLYLINE"
        Case sdfPolyPolylineObject:
            GetObjectTypeString = "POLYPOLYLINE"
        Case sdfPolyPolygonObject:
            GetObjectTypeString = "POLYPOLYGON"
    End Select

End Function
```

Index

A

Active Server Pages (ASP)
  about 79, 102
  creating reports with 102
  querying and updating data with 6
ActiveX Control version of the Autodesk MapGuide Viewer
  detecting with CODEBASE parameter 21
  displaying the map in 12
Allaire ColdFusion
  about 79
  creating reports with 55, 59, 62, 77, 78
  querying and updating data with 6, 77
annotations
  see redlining
API
  see Autodesk MapGuide Viewer API
ASP
  see Active Server Pages (ASP)
attribute data
  displaying for selected features (reports) 77, 78
  listing with ColdFusion 81, 83
  updating dynamically 78, 94, 119
Autodesk MapGuide Author
  creating popup menus with 100, 126
  creating reports with 83, 94, 108, 119
Autodesk MapGuide development
  introduction 5–8
Autodesk MapGuide LiteView Extension 14
Autodesk MapGuide SDF Component Toolkit 129–166
Autodesk MapGuide Server
  and SDF file access 134
  security 27
  URL of your 12, 14
Autodesk MapGuide Viewer
  ActiveX Control 12
  API
    accessing maps with 15
    accessing reports with 90, 101, 115, 127
    adding map layers 32
    advanced examples 49, 136
    autoRefresh flag 24
    busy state 23
Autodesk MapGuide Viewer API
  API (*continued*)
    counting map features 42
    counting map layers 30
    customizing the printout 44–49
    developing with 9–27, 29–76
    display refresh 23
    error checking 26
    events
      about 18
      example of 20
      handlers and observers 18
      Internet Explorer and 19
      Netscape Navigator and 19
    examples of common tasks 30
    linking map layers 33
    listing map layers 31
    radius mode 39
    retrieving feature coordinates 37
    retrieving feature keys 35
    security 26
    toggling map layer visibility 40
    zooming to selected features 41
  creating an application 11
  Java Edition 14
  LiteView Extension 14
  Plug-In 12

B

busy state, handling 23

C

CFM files 81, 83, 94
CODEBASE parameter
  detecting ActiveX Control with 21
ColdFusion
  about 79, 80
  creating reports with 55, 59, 62, 77, 78, 80
  listing database contents with 81, 83, 94
  querying and updating data with 6, 77
  template files 81, 83, 94
coordinates
  retrieving with the API 37
counting
  map features with the API 42
  map layers with the API 30

custom applications
creating with the Viewer API 9–27, 29–76

D

data
attribute
listing with ColdFusion 81, 83
updating dynamically 78, 94, 119
reports 77, 78
databases
listing contents with ColdFusion 81, 83, 94
updating via the map 78, 94, 119
debugging 26
digitizing
circles 39
points 53
display refresh, controlling 23
displaying the map 11
DSNs
working with 81, 94, 105, 108, 119
see also OLE DB

E

embedding MWFs in HTML pages 12
events, API 18

F

facilities management
sample application 69
files and directories
.asp 105, 108, 119
.cfm 81, 83, 94
.kif 133
.sdf 129–166
.sif 133

H

HTML pages
creating 82, 86, 95, 107, 111, 120

I

Internet Explorer
accessing map object from 15
and observer objects 17
embedding a map 13
event observers in 19–21
JavaScript support in 15
JavaScript/JScript support in 17
introduction to Autodesk MapGuide development 5–8

J

Java edition of the Autodesk MapGuide Viewer
API for the 17
displaying the map in 14
JavaScript
accessing reports with 90, 101, 115, 127
JavaScript/JScript support of Java edition 17

K

keys
retrieving with the API 35
KIF files 133

L

layers
adding with the API 32
counting with the API 30
linking with the API 33
listing with the API 31
toggling on and off with the API 40
legend
suppressing in printout 44
linking
map layers with the API 33
MWFs to HTML pages 12
listing map layers with the API 31
LiteView Extension of the Autodesk MapGuide Viewer 14

M

map features
accessing programmatically 15
counting 42
retrieving coordinates of 37
retrieving keys of 35
map legend
suppressing in printout 44
map title
customizing in printout 44
Map Window Files (MWFs)
accessing with the Autodesk MapGuide Viewer API 15
adding to HTML pages 11
Map Window Properties dialog box
Popup Menu tab 100, 126
Reports tab 94, 108, 119
MapGuideObserver5.class 20, 21, 138
maps
adding to HTML pages 11
customizing the printout 44–49
menus, adding items to 100, 126
Microsoft Active Server Pages
see Active Server Pages
Microsoft Internet Explorer
see Internet Explorer

N

Netscape Navigator
accessing map object from 15
embedding a map 13

Netscape Navigator (*continued*)
event observers in 19–21
JavaScript support in 17
north arrow
suppressing in printout 44

O

OLE DB
working with data sources 81, 94, 103, 105, 108, 119
see also DSNs
overview of Autodesk MapGuide development 5–8

P

page elements
adding 48
positioning 47
setting print priority for 47
Plug-In version of the Autodesk MapGuide Viewer
displaying the map in 12
popup menus, creating 100, 126
print coordinate system 47
printout
customizing with the API 44–49
enabling print events 44
handler functions for print events 44
setting priority of page elements 47

Q

querying data (reports) 77, 78

R

radius mode, invoking 39
redlining
compared to updating SDFs 136
sample application 49
refresh of the display, controlling 23
reports
about 78
accessing with JavaScript 90, 101, 115, 127
accessing with the Viewer API 90, 101, 115, 127
adding to a map 77, 78
and server-side scripting 77, 78
creating in Autodesk MapGuide Author 83, 94, 108, 119
creating with ASP 102
creating with ColdFusion 55, 59, 80
retrieving
coordinates of selected features with the API 37
keys of selected features with the API 35

S

sample code 9–27, 29–76, 81–127, 129–166
scale bar, suppressing in printout 44
SDF files
see Spatial Data Files (SDFs)
security
Autodesk MapGuide Viewer API and 26
server pages
see Active Server Pages (ASP)
server-side scripts and applications 77, 78
SIF files 133
Spatial Data Files (SDFs) 129–166
converting to 146
copying 161
editing 134
getting information about 153
indexing 133
pitfalls 134
updating via the map 136
using Visual Basic to work with 146
working with 133–135
support for JavaScript/JScript 17
symbols
adding to the printout 44

T

Toolkit
see Autodesk MapGuide SDF Component Toolkit

U

updating databases via the map 78, 94, 119
URL parameters, reports and 85, 95, 110, 120

V

Viewer API
see Autodesk MapGuide Viewer API

Z

zooming with the API 41

Despite rigorous product testing, some problems simply cannot be detected in advance. Let us know if you discover what may be a bug in our software. We'll address the problem, so that our software can take care of your business.

AUTODESK BUG REPORT

Instructions

1. Please fill in the form **completely**. Fill in the release number and serial number for your Autodesk product (Autodesk MapGuide Author, Autodesk MapGuide Server, etc.). Be sure to provide **ALL** the information about your system, as these specifics are important. For peripherals, specify actual make and model. If the peripheral is emulating another make or model, please note what that is. Please indicate all network information requested on this form.
2. Under **Problem Description**, describe the problem clearly and completely. We want to be able to re-create your problem, so we need to know the exact sequence of activities that led up to it. Include the exact error message, if one appeared. Use a separate sheet of paper if necessary. Please include information about programs, services, or utilities that are running but not a part of the native operating system.
3. If your problem concerns a particular drawing, please enclose a drawing disk. Attach any other relevant materials and check the corresponding boxes.
4. Mail to:

   Autodesk, Inc.
   111 McInnis Parkway
   San Rafael, CA 94903
   Attn: Bug Report

Address Information

Name _____ Company _____

Date _____ Address _____

Phone number _____

Extension _____ City _____ State _____
ZIP/Postal Code

Email address _____ Country _____

Complete this section only if you are an Authorized Dealer:

Your customer's name _____ Your customer's phone number _____

continued on back

Hardware and Software Information

Product Name

Serial Number

Computer Brand Name

Model

Operating System(s)/Version

Network Software/Version

Number of Nodes

Memory (Total RAM)

Hard Disk Space

Graphics Card(s)

Digitizer/Mouse

Plotter ☐ Serial ☐ Parallel

Printer ☐ Serial ☐ Parallel

Problem Description

Use this space to describe the problem. Be specific in the sequence of steps that led up to the problem and describe the exact results. Be sure to enclose copies of relevant materials: drawing files (on disk), script files, plots, etc.

Materials Enclosed

☐ Disk ☐ Script ☐ Letter ☐ Print/Plot/Image

We'd like to take credit for designing the world's finest software, but the truth is that much of the credit goes to you, our customer. If you have an idea for a new feature in the next release of one of our products, or hope to see an existing feature improved, please let us know.

AUTODESK WISHLIST

Address Information

Please send to:
Autodesk, Inc.
111 McInnis Parkway
San Rafael, CA 94903
Attn: Wish List
You can also submit wishlist items through the Autodesk Web page at *www.autodesk.com/wishlist*

Name _____ Company _____

Date _____ Address _____

Phone number _____ City _____

Extension _____ State _____ ZIP/Postal Code _____

Email address _____ Country _____

Complete this section only if you are an Authorized Dealer:

Your customer's name _____ Your customer's phone number _____

Please identify the Autodesk product this request is for:

_____

Please indicate the product release (or version) you are currently using:

Product _____ Serial Number _____ Platform/Operating System _____

Choose the category that best fits your request:

- ☐ New Feature or Command
- ☐ Feature or Command Enhancement
- ☐ Documentation Change
- ☐ Display Support
- ☐ Digitizer Support
- ☐ Operating System Support
- ☐ Printer/Plotter Support
- ☐ Platform Support
- ☐ Installation and Configuration
- ☐ Customization
- ☐ General
- ☐ Other _____

If applicable, indicate which feature or command this request relates to:

_____

continued on back

Describe your request in detail below (one request per sheet):

Please indicate the reasons for your request (i.e., what would be the benefits of your request and what problems would it solve?):

Thank You

We appreciate your interest in our products, and will consider your suggestions in our future product development.