

Autodesk MapGuide[®] Release 6

autodesk[®]

Developer's Guide

Copyright © 2001 Autodesk, Inc.

All Rights Reserved

AUTODESK, INC. MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, REGARDING THESE MATERIALS AND MAKES SUCH MATERIALS AVAILABLE SOLELY ON AN "AS-IS" BASIS.

IN NO EVENT SHALL AUTODESK, INC. BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF PURCHASE OR USE OF THESE MATERIALS. THE SOLE AND EXCLUSIVE LIABILITY TO AUTODESK, INC., REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE MATERIALS DESCRIBED HEREIN.

Autodesk, Inc. reserves the right to revise and improve its products as it sees fit. This publication describes the state of this product at the time of its publication, and may not reflect the product at all times in the future.

Autodesk Trademarks

The following are registered trademarks of Autodesk, Inc., in the USA and/or other countries: 3D Plan, 3D Props, 3D Studio, 3D Studio MAX, 3D Studio VIZ, 3DSurfer, ActiveShapes, ActiveShapes (logo), Actrix, ADE, ADI, Advanced Modeling Extension, AEC Authority (logo), AEC-X, AME, Animator Pro, Animator Studio, ATC, AUGI, AutoCAD, AutoCAD Data Extension, AutoCAD Development System, AutoCAD LT, AutoCAD Map, Autodesk, Autodesk Animator, Autodesk (logo), Autodesk MapGuide, Autodesk University, Autodesk View, Autodesk WalkThrough, Autodesk World, AutoLISP, AutoShade, AutoSketch, AutoSurf, AutoVision, Biped, bringing information down to earth, CAD Overlay, Character Studio, Design Companion, Design Your World, Design Your World (logo), Drafrix, Education by Design, Generic, Generic 3D Drafting, Generic CADD, Generic Software, Geodyssey, Heidi, HOOPS, Hyperwire, Inside Track, Kinetix, MaterialSpec, Mechanical Desktop, Multimedia Explorer, NAAUG, ObjectARX, Office Series, Opus, PeopleTracker, Physique, Planix, Powered with Autodesk Technology, Powered with Autodesk Technology (logo), RadioRay, Rastation, Softdesk, Softdesk (logo), Solution 3000, Texture Universe, The AEC Authority, The Auto Architect, TinkerTech, VISION*, *WHIP!*, *WHIP!* (logo), Woodbourne, WorkCenter, and World-Creating Toolkit.

The following are trademarks of Autodesk, Inc., in the USA and/or other countries: 3D on the PC, 3ds max, ACAD, Advanced User Interface, AME Link, Animation Partner, Animation Player, Animation Pro Player, A Studio in Every Computer, ATLAST, Auto-Architect, AutoCAD Architectural Desktop, AutoCAD Architectural Desktop Learning Assistance, AutoCAD Learning Assistance, AutoCAD LT Learning Assistance, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk Animator Clips, Autodesk Animator Theatre, Autodesk Device Interface, Autodesk Inventor, Autodesk PhotoEDIT, Autodesk Software Developer's Kit, Autodesk Streamline, Autodesk View DwgX, AutoFlix, AutoSnap, AutoTrack, Built with ObjectARX (logo), ClearScale, Colour Warper, Combustion, Concept Studio, Content Explorer, cornerStone Toolkit, Dancing Baby (image), Design 2000 (logo), DesignCenter, Design Doctor, Designer's Toolkit, DesignProf, DesignServer, DWG Linking, DXF, Extending the Design Team, FLI, FLIC, GDx Driver, Generic 3D, gmax, Heads-up Design, Home Series, i-drop, Kinetix (logo), ObjectDBX, onscreen onair online, Ooga-Chaka, Photo Landscape, Photoscape, Plasma, Plugs and Sockets, PolarSnap, Pro Landscape, Reactor, Real-Time Roto, Render Queue, SchoolBox, Simply Smarter Diagramming, SketchTools, Sparks, Suddenly Everything Clicks, Supportdesk, The Dancing Baby, Transform Ideas Into Reality, Visual LISP, Visual Syllabus, VIZable, Volo, and Where Design Connects.

Third Party Trademarks

Apple and Macintosh are trademarks of Apple Computer, Inc., registered in the U.S. and other countries.

ColdFusion is a registered trademark of Macromedia, Inc. All rights reserved.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Microsoft and ActiveX are registered trademarks of Microsoft Corporation in the United States and/or other countries.

All other brand names, product names or trademarks belong to their respective holders.

Third Party Software Program Credits

Copyright © 2001 Microsoft Corporation. All rights reserved.

Portions of this product are distributed under license from D.C. Micro Development, © Copyright D.C. Micro Development. All rights reserved.

InstallShield™ Copyright © 2001 InstallShield Software Corporation. All rights reserved.

GOVERNMENT USE

Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

Contents

Chapter 1	Introduction	7
	What's New in Release 6	8
	Autodesk DWG Data Source Support	8
	Enhanced Support for Map Redraw Operations	8
	Symbol Bitmap Support.	8
	Map Mode Retrieval Support	9
	Java Edition Platform Support Changes	9
	Before You Begin	9
	Familiarity with Autodesk MapGuide	9
	Programming and Scripting Languages	10
	Your Audience.	10
	User Help	11
	About the Autodesk MapGuide Viewer API	11
	Autodesk MapGuide Viewer API Help	13
	Autodesk MapGuide Web Site	13
	What Is an Autodesk MapGuide Viewer API Application	13
	Viewing Maps	14
	Querying and Updating Data	14
	Creating an Autodesk MapGuide Viewer API Application	15
Chapter 2	Displaying Maps	17
	Overview	18
	Map Display for ActiveX Control and Plug-In.	20
	Linking to a Map	20
	Embedding a Map	22
	Map Display for the Java Edition.	24
	Map Display for Autodesk MapGuide LiteView	26
	Installing Viewers on Client Machines	27
Chapter 3	Accessing Maps	31
	Overview	32
	Map Access for ActiveX Control and Plug-In	32
	Required Software for Autodesk MapGuide Viewer Plug-In.	34

Map Access for the Java Edition	35
Map Access from HTML.	35
Map Access Using Java	38
Required Software for the Java Edition	39
Java Edition Differences.	39
Communicating with the Plug-In from a Java Applet	40
Accessing Secure Data	43
Handling Busy State and Map Refresh	43
About the Busy State.	43
Controlling Map Refresh Operations.	45
Handling Errors	47
Getting Error Codes	47
Checking for Incorrect Argument Types	48
Debugging an Application	48
Chapter 4 Working with Map Layers, Map Features, and Printing	49
Overview	50
Working with Map Layers	50
Counting Map Layers	50
Listing Map Layers	51
Adding a Map Layer	53
Linking Map Layers	53
Toggling Map Layer Visibility On and Off	55
Working with Map Features	56
Getting Keys of Selected Map Features	56
Getting Coordinates of a Selected Map Feature	58
Invoking Select Radius Mode	61
Zooming In on Selected Features	61
Counting Map Features	62
Working with Printing	64
Setting the Print Priority	65
Enabling the Print Events	66
Positioning Page Elements with Page Coordinate System Units	67
Adding Custom Page Elements	68
Chapter 5 Handling Events	69
Overview	70
Working with Event Handlers.	70
Browser Differences	71
Setting Up Event Handlers.	72
Plug-In and Java Edition Event Handlers	72
ActiveX Control Event Handlers	75
Plug-In and ActiveX Control Event Handlers	75

Writing Event Handlers	78
Page Setup Event Handler Example	79
Print Event Handler Example	81
Plug-In Event Handler Example	83
Chapter 6 Using Reports to Query and Update Data Sources	85
Overview	86
How Reports Are Generated	86
Specifying the Report Script	86
The Request	87
Launching the Report	87
Introducing ColdFusion and ASP	88
Creating Report Scripts with ColdFusion	89
Listing File Contents with ColdFusion	89
Querying and Displaying Data via the Map with ColdFusion	92
Modifying a Database via the Map with ColdFusion	102
Creating Report Scripts with ASP	109
Summary of ASP Objects, Components, and Events	110
Listing File Contents with ASP	111
Querying and Displaying Data via the Map with ASP	115
Modifying a Database via the Map with ASP	125
Chapter 7 Applications	135
Overview	136
Custom Redlining Application	136
Redlining Example Code	137
Municipal Application	140
Municipal Application Example Code	141
Facility Management Application	156
Facilities Management Application Example Code	157
DWG Filtering Application	164
Understanding Layers in Autodesk MapGuide	165
Changing Map Layer Data Source Properties	165
DWG Filtering Application Example Code	166
SDF Component Toolkit Applications	171
Updating SDF Files—an ASP Example	171
Converting to an SDF File—a Visual Basic Example	181
Getting SDF File Information—a Visual Basic Example	188
Copying an SDF File—a Visual Basic Example	196
Index	203

Introduction

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 customization and development features. This chapter introduces Autodesk MapGuide application development, and describes how to use Autodesk MapGuide® Viewer API to develop such applications.

1

- What's new in Release 6
- Before you begin
- About the Autodesk MapGuide Viewer API
- What is an Autodesk MapGuide Viewer API application
- Creating an Autodesk MapGuide Viewer API application

What's New in Release 6

This release of the Autodesk MapGuide Viewer API includes one new object and several new and changed methods and properties to support new and enhanced features. For detailed information about the new object, and the methods and properties that were added or changed, choose [Help](#) ► [Contents](#) ► [What's New in the Autodesk MapGuide Viewer API Help](#).

Autodesk DWG Data Source Support

Several changes and additions have been made to Autodesk MapGuide and the Autodesk MapGuide Viewer API to support Autodesk drawing (DWG) data sources. The most significant addition is a new `MGDwgDataSources` object with several methods and properties to get and set DWG values of Autodesk MapGuide map layers. Also, a new `MGMapLayerSetup` method (`getDwgDataSources`) and a property (`MGDwgDataSources`) were added. Two `MGMapLayerSetup` methods (`getDataFile` and `setDataFile`) methods and one property (`DataFile`) were also changed to support DWG data sources. See “DWG Filtering Application” on page 164 for an example of working with DWG data sources.

Enhanced Support for Map Redraw Operations

Two new `MGMap` methods (`getIntermediateUpdatesEnabled` and `setIntermediateUpdatesEnabled`) and a new property (`IntermediateUpdatesEnabled`) were added to allow more control over how frequently, and under what conditions, the Autodesk MapGuide Viewer redraws a map. Maps are now updated every 1.5 seconds by default. This allows the end user to view changes to the map as it is being drawn instead of waiting until the entire map is rendered. See “Handling Busy State and Map Refresh” on page 43 for information about controlling map refresh operations.

Symbol Bitmap Support

Changes were made to two `MGSymbolAttr` methods (`setRotation` and `setSymbol`) and two properties (`Rotation` and `Symbol`) to support bitmaps in symbols.

Map Mode Retrieval Support

One new MGMap method (`getMode`) and one new property (`Mode`) were added to allow retrieval of a map's mode, such as zoom, pan, and so on.

Java Edition Platform Support Changes

Autodesk MapGuide Viewer, Java™ Edition does not run if you are using Microsoft® Internet Explorer® 5.x with Macintosh® Runtime for Java (MRJ) 2.2. You need to upgrade to MRJ 2.2.4 or later.

For a complete list of supported platforms, including browsers and operating systems, refer to “Autodesk MapGuide Viewer Requirements” in Chapter 1, “Introduction,” in the *Autodesk MapGuide User's Guide*.

Before You Begin

Before you can begin developing with Autodesk MapGuide, you need to know how to use Autodesk MapGuide and the programming and scripting languages you use to create Autodesk MapGuide web applications. Equally important, you need to know who your users are and what they need, as well as how to deliver help about your applications.

Familiarity with 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

You'll need to be familiar with one or more of the following programming languages or toolkits to develop Autodesk MapGuide web applications. MapGuide applications can include a variety of capabilities including customized interface (toolbar, etc.), report generation, processing of redlining markup, server-side processing of SDF files, Dynamic Map Authoring, and more.

- Use Java, JavaScript, JScript, Visual Basic, or VBScript with the Autodesk MapGuide Viewer API to develop applications that programmatically access and control Autodesk MapGuide Viewer. This document covers what you need to know to develop such applications.
- Use Macromedia® ColdFusion®, Microsoft® Active Server Pages™ (ASP), or another third-party application to create custom reports. For information about creating reports, see Chapter 6, “Using Reports to Query and Update Data Sources.”
- Use the SDF Component Toolkit to create server-side scripts that dynamically update SDF files posted on an Autodesk MapGuide® Server. For more information about using this toolkit, see “SDF Component Toolkit Applications” on page 171 and refer to the *SDF Component Toolkit Help*.
- Use the Autodesk MapGuide Viewer API to process redlining data and update your data sources. For more information about working with redlining data, see “Custom Redlining Application” on page 136.
- Use the Dynamic Authoring Toolkit to build XML-based applications for dynamic map solutions. For more information about this toolkit, refer to the *Dynamic Authoring Toolkit Developer's Guide*.

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? It's critical that you find out what tasks your users will need to perform, as well as their knowledge of those tasks.

User Help

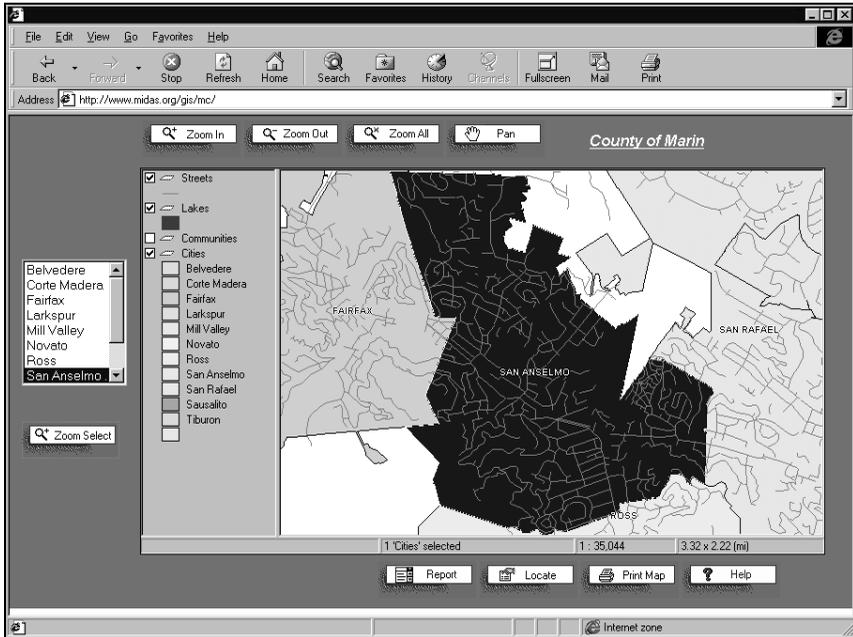
If you want to provide information about your application that users can readily access, you can develop your own set of Help pages. You can then set up the map to point to your customized Help system instead of the default *Autodesk MapGuide Viewer Help* when users click the Help button or access Help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

About the Autodesk MapGuide Viewer API

The Autodesk MapGuide Viewer API allows you to customize the way in which someone using Autodesk MapGuide Viewer interacts with a map. You can also create a stand-alone version of the Autodesk MapGuide Viewer that displays maps without the use of a Web browser.

Autodesk MapGuide Viewer comes in three different types: Autodesk MapGuide Viewer ActiveX Control, Autodesk MapGuide Viewer Plug-In, and Autodesk MapGuide Viewer, Java Edition. These Viewer types are sometimes referred to as Viewer versions within the Help and the documentation. Be careful to not confuse this usage with Autodesk MapGuide release versions. Each Autodesk MapGuide Viewer type exposes a programming interface that you can use to programmatically access and manipulate its functionality.

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 on the HTML page. 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 page allows you to create very powerful applications.

In this context, an Autodesk MapGuide Viewer application is a Web page containing one or more maps, each of which is displayed in a separate instance of the Autodesk MapGuide Viewer. The Web page can also have frames, buttons, controls, graphics, and so on for querying and controlling the map and its data. In most cases, you will write your application code within one or more HTML pages using one of the supported languages (refer to “Choosing a Viewer/Browser Environment” in the *Autodesk MapGuide User's Guide*).

Autodesk MapGuide Viewer API Help

For complete information about the Autodesk MapGuide Viewer API, refer to *Autodesk MapGuide Viewer API Help*, available from the Autodesk MapGuide CD and the Autodesk MapGuide documentation page at the following web site: www.autodesk.com/mapguidedocs. The *Autodesk MapGuide Viewer API Help* provides descriptions of all of the Autodesk MapGuide Viewer API objects, methods, properties, and events, and it includes sample applications that you can use to get a quick start.

Autodesk MapGuide Web Site

You can find additional information about Autodesk MapGuide development on the Autodesk MapGuide Web site at www.autodesk.com/mapguide. The site provides many examples of applications developed with Autodesk MapGuide, both demo applications and real customer sites. You will also find links to resources, such as *Autodesk MapGuide Viewer API Help*, API examples, general product documentation, and discussion groups.

What Is an Autodesk MapGuide Viewer API Application

An Autodesk MapGuide Viewer API 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.

The following sections describe several of the tasks your Autodesk MapGuide application can perform.

Viewing 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 Autodesk MapGuide Viewer runs within the user's Web browser to display the map, or you can run the Autodesk MapGuide Viewer from within a stand-alone application that you create. With either approach, you will use the Autodesk MapGuide 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.

Querying and Updating 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 Macromedia 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 of last service. For more information about reports, see Chapter 6, "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 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. For more information about redlining, see "Custom Redlining Application" on page 136. If your source data is in SDF files, you can use the SDF Component Toolkit to update the SDFs directly when redlining. For more information about working with SDF files, see "SDF Component Toolkit Applications" on page 171.

Creating an Autodesk MapGuide Viewer API Application

The following table summarizes tasks involved when creating an Autodesk MapGuide Viewer API application. For examples of real-world applications, also see Chapter 7, “Applications.”

Application Type	Task	For more information...
Simple Applications	Display maps by either linking to or embedding them in an HTML page	See Chapter 2, “Displaying Maps”
	Programmatically access and manipulate maps, map layers, and map features	See Chapter 3, “Accessing Maps” and Chapter 4, “Working with Map Layers, Map Features, and Printing”
Advanced Applications	Respond to Autodesk MapGuide Viewer events	See Chapter 5, “Handling Events”
	Query and update data sources using reports	See Chapter 6, “Using Reports to Query and Update Data Sources”

Displaying Maps

This chapter describes how to link to or embed a map in a Web page for display in Autodesk MapGuide® Viewer ActiveX Control, Autodesk MapGuide Viewer Plug-In, or Autodesk MapGuide Viewer, Java Edition. Once your application can display a map, you can access the map programmatically, as described in Chapter 3, “Accessing Maps.”

2

- Overview
- Map display for ActiveX Control and Plug-In
- Map display for the Java Edition
- Map display for Autodesk MapGuide LiteView
- Installing Viewers on client machines

Overview

The process of displaying a map involves linking to or embedding a specific map in a Web page. The Autodesk MapGuide Viewer installed on each user machine runs automatically to display the map. This means that your users can view the same map with one of three Autodesk MapGuide Viewers (Autodesk MapGuide Viewer ActiveX Control, Autodesk MapGuide Viewer Plug-In, or Autodesk MapGuide Viewer, Java Edition), depending on which ones you support. If your users are unable to download or install an Autodesk MapGuide Viewer, or if they do not need query or more advanced functionality provided by Autodesk MapGuide Viewers, they can use Autodesk MapGuide® LiteView instead. For more information about this viewing solution, see “Map Display for Autodesk MapGuide LiteView” on page 26.

The Autodesk MapGuide Viewers you choose to support depend on the browser operating environment of your users. The programming or scripting languages you use to develop your applications depend on the Autodesk MapGuide Viewers you support. For a detailed discussion of these options, refer to “Choosing a Viewer/Browser Environment” in the *Autodesk MapGuide User’s Guide*.

The following table summarizes the combinations supported by Autodesk MapGuide.

Browser/Viewer Configurations

Operating System	Browser	Client-Side Viewer	Programming or Scripting Language
Windows	Internet Explorer	Autodesk MapGuide Viewer ActiveX Control	HTML, VBScript, JScript, JavaScript
		Autodesk MapGuide Viewer, Java Edition	HTML, JScript, JavaScript, Java
	Netscape Navigator	Autodesk MapGuide Viewer Plug-In	HTML, JavaScript
		Autodesk MapGuide Viewer, Java Edition	HTML, JavaScript, Java
	None (stand-alone application)	Autodesk MapGuide Viewer ActiveX Control	Visual Basic
Mac OS	Internet Explorer	Autodesk MapGuide Viewer, Java Edition	HTML, Java
Solaris	Netscape Navigator	Autodesk MapGuide Viewer, Java Edition	HTML, JavaScript, Java

Map Display for ActiveX Control and Plug-In

To display your map with Autodesk MapGuide Viewer ActiveX Control and/or Autodesk MapGuide Viewer Plug-In, you can either:

- Link to the map, or
- Embed the map (the map runs inside a Web browser)

Linking to a Map

You can display a map by creating a link to the map from an HTML page. The, when the user clicks the link, the map displays full screen. Note that the browser displays the map by itself, not as part of an HTML page.

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 file for the map:

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



Linked map

Displaying a Linked Map in a Different Window or Frame

To display the linked map in a different window or frame, use the `TARGET` parameter. If necessary, open the second window or frame so that the original document can continue to be displayed. For example:

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

This displays the *usa.mwf* map in a Web browser window called `MAPAREA`. The `TARGET` parameter can also specify the name of a frame.

Displaying a Specific Area of the Map

In an Autodesk MapGuide Viewer, you can adjust the view of the map window so that it displays only the area you want.

To display a specific area of a map

- 1 Right-click over the map to display the popup menu, and then use the Zoom commands or Pan to display the area of the map you want.
- 2 Right-click again, and then choose Copy ► As URL to copy the map's URL to the clipboard.
- 3 In your HTML document, choose Edit ► Paste to paste the URL into the `HREF` parameter of the anchor tag. For example:

```
<A HREF="http://www.mapguide.com/maps/usa.mwf?Lat=37.81  
&Lon=-122.37&Width=20.0&Units=M&ext=.mwf">United States Map</A>
```

This displays the San Francisco area. For a list of the parameters that control the way the map is displayed when linked to or embedded in an HTML page, choose Help ► Contents ► Advanced Topics ► URL Parameters in the *Autodesk MapGuide Viewer Help*.

Embedding a Map

A second way to display a map is to embed it in an HTML page. Embedding a map displays it with the rest of the information on that page.

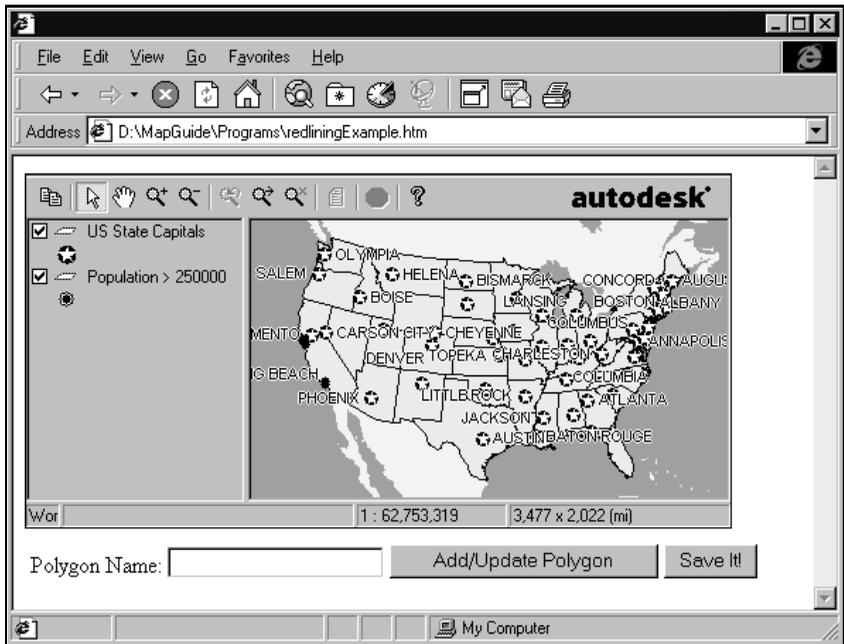
To embed the map, use the `EMBED` (for Netscape Navigator) or `OBJECT` (for Microsoft Internet Explorer) tag in the page.

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

```
<OBJECT ID="map" 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="map"  
    WIDTH=300 HEIGHT=200>  
</OBJECT>
```

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, choose [Help](#) ► [Contents](#) ► [Advanced Topics](#) ► [URL Parameters](#) in the *Autodesk MapGuide Viewer Help*. Be sure that the values you use are the same for both the `OBJECT` and `EMBED` parameters.

The following screen shows the embedded map.



Embedded map

Embedding a Map in a Frame-Based Page

To display a map within a frame-based page, use the `<FRAME>` tag to reference an HTML document that embeds the map. When you display a map within a frame, the size of the map can grow and shrink as the window is resized. To see an example of a map within a simple frame-based HTML page, choose **Help** ► **Contents** ► **Examples Basic** ► **Simple Frameset example** in the *Autodesk MapGuide Viewer API Help*.

The following is an example of an HTML page for frame set layout:

```
<HTML>
<HEAD>
<TITLE>Frame Layout</TITLE>
</HEAD>
  <FRAMESET ROWS="70%,*">
    <FRAME SRC="map.htm" NAME="myFrame" SCROLLING=no
      MARGINHEIGHT=0 MARGINWIDTH=0>
    <FRAME SRC="list.htm" NAME="ListFrame">
  </FRAMESET>
</HTML>
```

Here is an example HTML page for *map.htm* referenced by the frame set:

```
<HTML>
<HEAD>
<TITLE>Map.htm</TITLE>
</HEAD>
<BODY>
<OBJECT ID="map" WIDTH=100% HEIGHT=100%
  CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D"
  CODEBASE="ftp://adeskftp.autodesk.com/webpub/mapguide/ver6/
    mgaxctrl.cab#Version=6,0,x,x">
  <PARAM NAME="URL" VALUE="http://www.mapguide.com/maps/usa.mwf">
  <PARAM NAME="ToolBar" VALUE="OFF">
  <EMBED SRC="http://www.mapguide.com/maps/usa.mwf?ToolBar=OFF"
    NAME="map" WIDTH=100% HEIGHT=100% BORDER=0>
</OBJECT>
</BODY>
</HTML>
```

Note that in your applications, you should change the release (`#Version`) number from `6,0,x,x` to the actual release number of Autodesk MapGuide you are using. You can find the release number using the Help ► Help About menu command in the user interface or the `MGMap.aboutDlg` method.

Map Display for the Java Edition

Using Autodesk MapGuide Viewer, Java Edition, you cannot display a map by linking to it; you must embed it. The following are ways to embed a map:

- Embed Autodesk MapGuide Viewer, Java Edition in the HTML page where the map is embedded (runs inside a Web browser).
- Wrap a Java applet around Autodesk MapGuide Viewer, Java Edition (can run inside or outside of a Web browser). For information about applet wrappers, see “Map Access Using Java” on page 38.

If you are using Autodesk MapGuide Viewer, Java Edition, you cannot link to a map. Instead, you must embed the map in the HTML page. To do this, use the `<APPLET>` tag and be sure to enter it directly into the HTML. Do not use `document.write` statements. For example, you should use:

```
<APPLET NAME="map" WIDTH=300...
```

and so on, and not use:

```
document.write('<APPLET');
document.write(' NAME=map');
...
```

Note Do not specify the height and width parameters in percentages, because percentages are unsupported in Internet Explorer on the Macintosh.

To display your map with Autodesk MapGuide Viewer, Java Edition

- 1 Start with the standard HTML `<APPLET>` tag.
- 2 Set the `CODE` parameter to the path to the `MGMapApplet.class` file in the `com.autodesk.mgjava` package. This package is where the API for Autodesk MapGuide Viewer, Java Edition resides. You can find this package in the `mgjava.jar` archive file, which contains all Autodesk MapGuide Viewer, Java Edition class files for Windows, Macintosh, or Solaris systems. There is also an `mgjava.cab` file for Windows only. To download these files, choose **Help** ► **Contents** ► **Plug-In/Java Edition Downloads** in the *Autodesk MapGuide Viewer API Help*. The `MGMapApplet` object provided by the `MGMapApplet.class` file implements the `MGMap` object interface for Autodesk MapGuide Viewer, Java Edition. Setting the `CODE` parameter to `MGApplet` instantiates Autodesk MapGuide Viewer, Java Edition.
- 3 Set the `VALUE` parameter to the URL of your Autodesk MapGuide Server and the path to the MWF file for the map. For example:

```
<HTML>
<HEAD>
<TITLE> Autodesk MapGuide Viewer, 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="mwfURL"
        VALUE="http://www.mapguide.com/maps/usa.mwf">
    </APPLET>
  </CENTER>
</BODY>
</HTML>
```

Map Display for Autodesk MapGuide LiteView

Autodesk MapGuide LiteView is a platform-independent, server-side viewing solution that delivers maps in the form of static raster images to client Web browsers from your Web page. It is not a type of Autodesk MapGuide Viewer. Autodesk MapGuide LiteView is useful when users need only to display the map and do not need the more advanced queries and other functionality of Autodesk MapGuide Viewer. Also, since LiteView is a server-side solution, users do not have to download and install one of the Autodesk MapGuide Viewer types to view your maps.

To display a map using Autodesk MapGuide LiteView, 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 Autodesk MapGuide Viewer API with Autodesk MapGuide LiteView. For complete information on implementing Autodesk MapGuide LiteView, refer to the *Autodesk MapGuide LiteView Developer's Guide*. The following table summarizes the supported configurations for Autodesk MapGuide LiteView:

Autodesk MapGuide LiteView Configurations

Operating System	Browser	Server-Side Viewing Solution	Programming or Scripting Language
Windows	Any browser that supports PNG file format	Autodesk MapGuide LiteView	ColdFusion (CF), Active Server Pages (ASP), Java Server Pages (JSP) or Perl
Mac OS			
Solaris			

Installing Viewers on Client Machines

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. You can include code in your HTML file that automatically detects whether or not the user has Autodesk MapGuide Viewer, and then either downloads it automatically or prompts the user to download it themselves.

To install the Autodesk MapGuide Viewer ActiveX Control

- 1 To install the latest version of Autodesk MapGuide Viewer ActiveX Control for Internet Explorer users, include the CODEBASE parameter to access the ActiveX Control cabinet file, *mgaxctrl.cab*, in the HTML page that specifies your map. This parameter detects whether the latest version of the Autodesk MapGuide Viewer ActiveX Control is installed. If the user does not have the ActiveX Control or has an older version, the latest version will be installed automatically.

In the following example, the CODEBASE parameter references the *mgaxctrl.cab* file located on the Autodesk MapGuide FTP site. Alternatively, you can copy this file to your Web server and reference the file there. You can find a copy of this file in the `\ActiveXCab` folder on the Autodesk MapGuide CD:

```
<OBJECT ID="map" WIDTH=300 HEIGHT=200
  CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D"
  CODEBASE="ftp://adeskftp.autodesk.com/webpub/mapguide/ver6/
  mgaxctrl.cab#Version=6,0,x,x">
<PARAM NAME="URL"
  VALUE="http://www.mapguide.com/maps/usa.mwf">
<EMBED SRC="http://www.mapguide.com/maps/usa.mwf"
  NAME="map" WIDTH=300 HEIGHT=200
  PLUGINSPAGE="<www.autodesk.com/mapguideviewerdownload>">
</OBJECT>
```

Note that in your applications, you should change the release (`#Version`) number from `6,0,x,x` to the actual release number of Autodesk MapGuide you are using. You can find the release number using the Help ► Help About menu command in the user interface or the `MGMap.aboutDlg` method.

To install the Autodesk MapGuide Viewer Plug-In

- 1 To install the latest version of Autodesk MapGuide Viewer Plug-In for Netscape Navigator users, write additional code to prompt the user to download Autodesk MapGuide Viewer Plug-In, as follows:

To download Autodesk MapGuide Viewer Plug-In

```
// Call this function on a page onLoad event or frameset onLoad event
function init()
{
    // For Netscape browsers, check for
    // Autodesk MapGuide Viewer Plug-In
    if (navigator.appName == "Netscape")
    {
        for(j=0;j<navigator.plugins.length;j++)
        {
            if (navigator.plugins[j].name == "Autodesk MapGuide")
                return;
        }
        // If the Autodesk MapGuide Viewer Plug-In is
        // not detected, display the message...
        displayDownloadMsg();
        return;
    }
    // If the Autodesk MapGuide Viewer Plug-In is installed,
    // check the version by returning the API version
    var version = getMap().getApiVersion();
    // If the API/Plug-In version is previous to 6.0,
    //display the message
    if (version < "6.0")
    {
        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 Autodesk MapGuide Viewer."
    // If user clicks OK, load download page from Autodesk Web site
    if (confirm(msg))
        top.window.location =
            "www.autodesk.com/mapguideviewerdownload";
}
```

To install the Autodesk MapGuide Viewer, Java Edition

- 1 To check a client machine for Autodesk MapGuide Viewer, Java Edition, include the following applet in the HTML page that displays your map:

```
<APPLET  
  WIDTH="200"  
  HEIGHT="50"  
  CODE="CheckInstall.class"  
  <PARAM NAME="ClassToCheck"  
    VALUE="com.autodesk.mgjava.MGMapApplet">  
  <PARAM NAME="Installer"  
    VALUE="http://www.autodesk.com/mapguideviewerdownload">  
</APPLET>
```

If Autodesk MapGuide Viewer, Java Edition is not found, the user is given the option to download it from the Autodesk MapGuide download page. For this code to work, the *CheckInstall.class* file must reside in the same directory as the Web page that contains this code. Alternatively, you can add a CODEBASE parameter that points to the folder where this file exists.

For a working example, choose Help ► Examples Advanced ► Java Edition Examples ► Example3 in the *Autodesk MapGuide Viewer API Help*. For a default Windows installation, you can find a copy of the *CheckInstall.class* file in the following location:

```
C:\Program Files\Autodesk\MapGuideDocumentation6  
  \ViewerAPIHelp\ViewerAPI\JavaEdition\examples\classes
```


Accessing Maps

This chapter describes how to programmatically access maps you have embedded in a Web page. Once your application can access a map, you can begin to expand your application to work with map layers, map features, and printing as described in Chapter 4, “Working with Map Layers, Map Features, and Printing.”

3

- Overview
- Map Access for ActiveX Control and Plug-In
- Map Access for the Java Edition
- Communicating with the Plug-In from a Java applet
- Accessing secure data
- Handling busy state and map refresh
- Handling errors
- Debugging an application

Overview

Netscape® Navigator® and Microsoft® Internet Explorer® expose map objects at different levels in their object hierarchies. Therefore, how you access a map is determined by the browsers you are supporting, not by the version of Autodesk MapGuide® Viewer you use. As described in this chapter, you can write a simple function that checks the user's browser type and returns an instance of the `MGMMap` object using the technique required by that browser. Note that we refer to JavaScript code modules as *functions*, reserving the term *method* for the Autodesk MapGuide Viewer API. Once you've obtained the map object, your application code uses that instance when it calls methods in the Autodesk MapGuide Viewer API. In most cases, the code for either browser will be identical.

Map Access for ActiveX Control and Plug-In

This section describes how to access maps from both Netscape Navigator and Internet Explorer. The techniques described here use JavaScript as the scripting language. 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 `map`. In Netscape Navigator, the map object called `map` is exposed by the document object and can be accessed from JavaScript in one of the following ways:

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

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

```
window.map           // one way...
map                  // another way...
```

The easiest way get around these differences is to write a `getMap` 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 had the name `map`, the code to access the map on a simple, frameless HTML page would look like this:

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

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

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

Note We chose the name `getMap` for our function, but the name can be anything you want, as long as it follows JavaScript naming conventions. Be careful to not to confuse the `getMap` function with `MGMapLayer.getMap`, a predefined Autodesk MapGuide Viewer API method.

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

```
var map = getMap();
```

You can 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 = getMap(); // assign map to variable
    jb_map.viewReport('Parcel Data'); // call method from variable
}
```

Or you could bypass the variable assignment and use `getMap` directly:

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

Because `getMap` 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.

Warning Because Netscape Communicator 4.x contains the Same Origin security policy, the `map` object cannot be accessed by the Autodesk MapGuide Viewer API code in a different frame. This occurs only when the code is written in Java or JavaScript and is on a different domain from the one where the map window file (MWF) resides.

To enable codebase principals

- 1 Make sure Netscape Communicator is not running.
- 2 In your `prefs.js` file, located in the `Netscape/user` directory, add the line:

```
user_pref("signed.Java.editions.codebase_principal_support", true);
```

- 3 In your JavaScript code, before making any Autodesk MapGuide Viewer API calls or references, add the line:

```
netscape.security.PrivilegeManager.enablePrivilege("UniversalBrowserRead");
```

- 4 When executing the JavaScript code, Netscape Communicator will ask whether to Grant or Deny access to the script. If you select Grant, the script will run normally. Otherwise, the script is prevented from executing.

Because codebase principals offer a minimal level of security, they can be useful during development of your code, but you should use object signing and digital signatures before delivery. For detailed information on Netscape Communicator's security models, codebase principals, and object signing, refer to the Netscape developer documentation.

Required Software for Autodesk MapGuide Viewer Plug-In

To work with Autodesk MapGuide Viewer Plug-In, you may need to download certain Java files provided with Autodesk MapGuide. Specifically, if your application will handle events, you will need to download the Java observer applet file `MapGuideObserver6.class`. For information about event handling with Autodesk MapGuide Viewer Plug-In, see Chapter 5, "Handling Events."

To download this observer applet file, choose Help ► Contents ► Plug-In/Java Edition Downloads in the *Autodesk MapGuide Viewer API Help*.

Map Access for the Java Edition

You can access maps using Autodesk MapGuide Viewer, Java Edition in two ways:

- From an HTML page by embedding Autodesk MapGuide Viewer, Java Edition in the HTML page where the map is embedded. This approach is described in “Map Display for the Java Edition” on page 24. In this case, Autodesk MapGuide Viewer, Java Edition runs inside a Web browser.
- Using Java by wrapping a Java applet around Autodesk MapGuide Viewer, Java Edition as described in “Map Access Using Java” on page 38. Using this approach, Autodesk MapGuide Viewer, Java Edition can run inside or outside a Web browser.

Map Access from HTML

In an HTML page, you can simply embed Autodesk MapGuide Viewer, Java Edition using the `<Applet>` tag. This approach provides the following benefits:

- You can post a map that most browsers and operating systems can access.
- This is the simplest way to support Autodesk MapGuide Viewer, Java Edition. All you have to do is use the standard HTML `<Applet>` tag and set `MGMapApplet` as the code value.

Although this approach supports the greatest number of browser/platform combinations, just displaying the map might not provide enough functionality for you. In this case, you might want to use JavaScript or JScript.

Using JavaScript or JScript

The JavaScript or JScript techniques for accessing maps described in “Map Access for ActiveX Control and Plug-In” on page 32 also work for Autodesk MapGuide Viewer, Java Edition since the object hierarchy is determined by the browser, not by the version of the Autodesk MapGuide Viewer.

Using JavaScript or JScript within an HTML page, you can create controls on the HTML page, such as list boxes and buttons, that interact with the map. However, JavaScript and JScript do not behave uniformly across all browser/operating system combinations. 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, the `digitizePoint` method requires an instance of the `MGDigitizePointObserver` object. Therefore, Internet Explorer would not be able to access `digitizePoint` or any other methods that pass observer objects as arguments, including the following `MGMap` methods. If you need to use any of these methods, implement Autodesk MapGuide Viewer, Java Edition from Java instead of JavaScript or JScript:
 - `addMapLayer` and `addMapLayers`
 - all of the `digitize` methods
 - `viewDistance` and `viewDistanceEx`

The following table lists the levels of support that JavaScript or JScript provide across different configurations.

JavaScript/JScript Support for Different Configurations

Operating System	Viewer Version	JavaScript or JScript Support
Windows	Autodesk MapGuide Viewer Plug-In on Netscape Navigator	Full support.
	Autodesk MapGuide Viewer ActiveX Control on Internet Explorer	
	Autodesk MapGuide Viewer, Java Edition on Netscape Navigator	
	Autodesk MapGuide Viewer, Java Edition on Internet Explorer	Only methods that do not pass observer objects as arguments.
Solaris	Autodesk MapGuide Viewer, Java Edition on Netscape	Full support.
Mac OS	Autodesk MapGuide Viewer, Java Edition on Internet Explorer	Internet Explorer 4.0 does not sup- port JavaScript and JScript cannot control Java applets.

Therefore, if you need to support the largest number of browsers and operating systems, you should embed the Autodesk MapGuide Viewer, Java Edition in an HTML page and not use either JavaScript or JScript to interact with the map. If you need more interactivity with the map, you will need to use Java instead, as described below. This is particularly important if you need to support Macintosh users, as they will need to use the Autodesk MapGuide Viewer, Java Edition with Internet Explorer, which doesn't have full JScript support.

Map Access Using Java

You can access maps using Java and the Autodesk MapGuide Viewer, Java Edition in several different ways, including using a wrapper Java applet, a peer Java applet, or a wrapper Java application.

Wrapper Java Applet

You can write a wrapper Java applet that uses the pre-installed Autodesk MapGuide Viewer, Java Edition and contains all the Autodesk MapGuide Viewer API methods. This approach enables the Autodesk MapGuide Viewer API to work on all operating systems and browsers. To run Autodesk MapGuide Viewer from a wrapper applet, code your wrapper to instantiate `MGMapComponent`. Then, in HTML, set the `CODE` value of the `<APPLET>` tag to the name of the wrapper applet. For a Java wrapper applet example, choose [Help > Contents > Examples Advanced > Java Edition Examples > Example6](#) in the *Autodesk MapGuide Viewer API Help*.

Peer Java Applet

You can write a peer Java applet that runs Autodesk MapGuide Viewer, Java Edition. This approach is slightly simpler than driving Autodesk MapGuide Viewer, Java Edition from a wrapper applet, because, with a peer java applet, Autodesk MapGuide Viewer, Java Edition is automatically provided with browser services such as `showDocument`. Another benefit is that Autodesk MapGuide Viewer, Java Edition and the peer applet can exist in separate frames in the HTML document. To run Autodesk MapGuide Viewer from a peer applet, your HTML page needs to include the `<APPLET>` tag twice: to set the `CODE` value to `MGMapApplet`, and to set it to the name of the peer applet.

Wrapper Java Application

You can write a Java wrapper application that runs Autodesk MapGuide Viewer, Java Edition outside a browser. This approach is analogous to using Visual Basic to run Autodesk MapGuide Viewer ActiveX Control outside a browser. You can still have Internet access from a Java application outside a browser, but browser-specific functionality will not be available. To run Autodesk MapGuide Viewer from a wrapper application, the application must embed the `MGMapComponent` object and then implement the `MGMapContext` object interface so that `MGMapComponent` can use the methods defined in `MGMapContext` that are normally provided by the browser.

Required Software for the Java Edition

To work with Autodesk MapGuide Viewer, Java Edition, you may need to download certain Java files provided with Autodesk MapGuide. Specifically, you will need to access the Autodesk MapGuide Viewer, Java Edition API, which resides in the *com.autodesk.mgjava* package. You can find this package in the *mgjava.jar* archive file, which contains all Autodesk MapGuide Viewer, Java Edition class files for Windows, Macintosh, or Solaris systems. There is also an *mgjava.cab* file for Windows only. To download these files, choose Help ► Contents ► Plug-In/Java Edition Downloads in the *Autodesk MapGuide Viewer API Help*.

If you are developing a Java application with Autodesk MapGuide Viewer, Java Edition, you will also need the Java Development Kit version 1.1.x. If you do not have this version, you can get it by downloading it from the following Web site: *java.sun.com*. If you are developing a Java application with Autodesk MapGuide Viewer, Java Edition using a development environment such as WebGain® VisualCafe™ or Borland® JBuilder, be sure to include the *mgjava.jar* file in your classpath.

If your application will handle events, you will need to download the Java observer applet file *MapGuideObserver6J.class*. For more information about event handling with Autodesk MapGuide Viewer, Java Edition, see Chapter 5, “Handling Events.” To download the Autodesk MapGuide observer, files, choose Help ► Contents ► Plug-In/Java Edition Downloads in the *Autodesk MapGuide Viewer API Help*.

Java Edition Differences

The following sections describe some functional and browser differences between Autodesk MapGuide Viewer, Java Edition and Autodesk MapGuide Viewer ActiveX Control and Autodesk MapGuide Viewer Plug-In.

Functional Differences with Java Edition

Bookmarks and map preferences are stored in a different location for Autodesk MapGuide Viewer, Java Edition than they are for Autodesk MapGuide Viewer ActiveX Control and Autodesk MapGuide Viewer Plug-In. They are not stored in the system registry. Instead, they are stored in the user’s local file system, relative to the user’s home directory at *\$home/.autodesk/mapguide/6.0* where *\$home* is the value of the Java system property `user.home`.

Different browsers and development environments have different assumptions about where the home directory is located, for example:

- **Internet Explorer**

`C:\windows_dir\Java\autodesk\mapguide\6.0`

- **Netscape**

`C:\Programs\Netscape\users\username\.autodesk\mapguide\6.0`

- **Solaris**

`/home/username/.autodesk/mapguide/6.0`

Browser Differences with Java Edition

- If a user with Internet Explorer requests data beyond a firewall, Autodesk MapGuide Viewer, Java Edition fails unless the browser has already been through proxy authorization. A simple workaround for this is to set the browser's home page to a site outside the firewall.
- If a map was authored so that double-clicking a feature runs a JavaScript command instead of opening an HTML page, the command might fail, depending on the browser. For example, if the URL for a map feature is set to:

```
javascript:alert("This is a private residence.")
```

users will receive an error message when they double-click that map feature in Internet Explorer. This is because different browsers support different protocols in the Java network library (the previous example illustrates that Internet Explorer does not support the JavaScript protocol in the Java network library).

Communicating with the Plug-In from a Java Applet

Netscape's LiveConnect technology makes it possible to write a Java applet that communicates with the Autodesk MapGuide Viewer Plug-In API. If you choose to develop your application this way, you will need to know how to:

- Access Autodesk MapGuide Viewer Plug-In from a Java applet
- Call JavaScript functions from an Applet

The following sections cover these topics. You may also need to handle events for Autodesk MapGuide Viewer Plug-In from your applet. For more information about how to do this, see Chapter 5, "Handling Events."

Accessing the Plug-In from a Java Applet

You can access Autodesk MapGuide Viewer Plug-In from your Java applet.

To access Autodesk MapGuide Viewer Plug-In

- 1 Copy the *npmav32.zip* file installed in your Netscape *Plugins* folder to a folder in your Java development environment, so that your Java compiler can import Autodesk MapGuide classes to your Java applet.
- 2 Write a `setMap` function in your applet that takes a parameter referencing the map and then storing the reference for future use. For example:

```
Private MMap myUSAMap;  
Public void setMap (MMap map1)  
{  
    myUSAMap = map1;  
}
```

- 3 Add your applet to the HTML page. You must include the `MAYSCRIPT` attribute in the `APPLET` tag to give the applet permission to interact with JavaScript. For example:

```
<APPLET code="plugInApplet" NAME="obs" WIDTH=400 HEIGHT=300  
MAYSCRIPT>  
.  
.  
.  
</APPLET>
```

- 4 If you have not already done so, embed Autodesk MapGuide Viewer Plug-In in the HTML page as shown in the following example (for more information, see "Embedding a Map," on page 22). Use the following code:

```
<EMBED  
SRC="http://www.mapguide.com/maps/usa.mwf?STATUSBAR=OFF"  
NAME="map" WIDTH=300 HEIGHT=200 BORDER=0>
```

- 5 Enter the following code into your HTML page:

```
<SCRIPT LANGUAGE="JavaScript">  
  
function Init()  
{  
    var map = document.map;  
    var obs = document.obs;  
    obs.setMap(map);  
}  
</SCRIPT>  
<BODY onLoad="Init();" >  
.  
.  
.  
</BODY>
```

This code calls the `Init` function, which is placed within the `onLoad` event, which is inserted into the `BODY` tag. This prevents errors by loading the map before any code tries to access it. `Init` passes the embedded map object into the `setMap` function in the applet. Now you can make any API call to the example file, `myUSAMap`, in your applet. For instance:

```
Public void getViewerInfo()  
{  
    myUSAMap.aboutDlg();  
}
```

The API is available from Autodesk MapGuide classes in the `npmav32.zip` file you copied in Step 1.

Calling JavaScript Functions from a Java Applet

You can call JavaScript functions in your HTML page from a Java applet.

To access JavaScript from a Java Applet

- 1 Import the `netscape.javascript` package into your Java applet code. You can usually find this package in the `\Program\Java\Classes\java40.jar` file in the Netscape browser root directory. You can download this file from <http://developer.netscape.com/software/jdk/download.html>. This package provides access to the `JSObject` class containing the `getWindow` method, which enables you to call JavaScript methods from your applet.
- 2 The syntax for calling a JavaScript function from your applet is:

```
JSObject.getWindow(applet instance).call("function name",  
parameters)
```

If you want to forward the `onDigitizedCircle` event to a JavaScript function called `onDigitizedCircleHandler`, you must first put all of the parameters into a `java.lang.Object` array before you can call `onDigitizedCircleHandler`. Since the fourth parameter of the `onDigitizedCircle` event is not derived from the `java.lang.Object` class, you need to wrap it as a `java.lang.Double` object:

```
public void onDigitizedCircle(MGMap map, String units, MGPoint  
center, double radius)  
{  
    Object[] params = new Object[4];  
    params[0] = map;  
    params[1] = units;  
    params[2] = center;  
    params[3] = new Double(radius);  
    JSObject.getWindow(this).call("onDigitizedCircleHandler",  
params);  
}
```

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.

Handling Busy State and Map Refresh

This section describes the map busy state and the techniques for coordinating when and how to refresh a displayed map. When Autodesk MapGuide Viewer refreshes the map 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 Autodesk MapGuide Viewer API is designed so you can understand how to code your application correctly.

About the Busy State

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

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

- In general, most write methods and properties are affected by the busy state and return a `-1` (`busy`) error code when called during a busy state. Most read methods and properties are not affected by the busy state. However, there are exceptions to both of these rules. The best way to learn which methods and properties do not work during the busy state is to view the method/property descriptions in the *Autodesk MapGuide Viewer API Help*.
- When your Autodesk MapGuide Viewer application calls an API method that causes a busy state, Autodesk MapGuide Viewer can return control to the application and then go to the next method while still in 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 the methods that are affected by the busy state.

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

```
function selectAndZoomToPointObject (mgObj)
{
    var map = getMap();
    var sel = map.getSelection();
    sel.clear();
    sel.addObject (mgObj);
    map.zoomSelected(); // Busy state begins in zoomSelected()
    map.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 map refresh using the `autoRefresh` flag
- Detect when a map refresh is about to happen
- Detect a change in the busy state

Each of these approaches is described in the following sections.

Controlling Map Refresh Operations

You can ensure that Autodesk MapGuide 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:

- When the `autoRefresh` flag is set to `True` with `MGMap.setAutoRefresh`, *and* the application calls an API method that requires an automatic refresh such as `MGMap.zoomSelected`. Methods requiring an automatic refresh are noted in the *Autodesk MapGuide Viewer API Help*.
- Your application calls `zoomGotoDlg`, `zoomGotoLocation`, `setUrl`, or `refresh`. These methods *always* invoke a display refresh, even if the `autoRefresh` flag is set to `false`. The only exception is if you call one of these methods from `onViewChanging` or `onMapLoaded` event handling code. Under these circumstances, the methods will fail and will set error code to `-14` (`refresh disabled`) because the `onViewChanging` and `onMapLoaded` events always disable the `autoRefresh` flag.

Using the autoRefresh Flag

To develop an application that executes smoothly, you need to prevent busy states while the application calls methods in the API that do not work during the busy state. To do this, you need to disable the `autoRefresh` flag.

To disable the autoRefresh Flag

- 1 Set the `autoRefresh` flag to `False` immediately before calling the first method.
- 2 Reset the `autoRefresh` flag to `True`.
- 3 Call the `refresh` method immediately after your application calls the other methods that do not work during the busy state. Simply setting the `autoRefresh` flag back to `True` with `MGMap.setAutoRefresh` does not refresh the map. For example:

```
function selectAndZoomToPointObject(mgObj)
{
    var map = getMap();
    var sel = map.getSelection();
    map.setAutoRefresh(false); // Prevent busy state from happening
                                // when zoomSelected is called

    sel.clear();
    sel.addObject(mgObj);
    map.zoomSelected();
    map.setWidth(5, "KM");
    map.setAutoRefresh(true); // Reset the autoRefresh flag
    map.refresh();           // Update the display
}
```

AutoRefresh Flag Caveats

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 may 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 may not appear in the printout or the buffer.

Detecting Map Refreshes

Autodesk MapGuide Viewer fires the `onViewChanging` and `onViewChanged` events both when a map 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 Chapter 5, “Handling Events”). However, before the Autodesk MapGuide 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

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.

Controlling Intermediate Update Map Redraw Operations

You can use the `MGMap.getIntermediateUpdatesEnabled` and `MGMap.setIntermediateUpdatesEnabled` methods to control how frequently Autodesk MapGuide Viewer redraws a map. By default, maps are updated every 1.5 seconds, enabling the end user to progressively view a map as it is being rendered, rather than waiting until the entire map is complete. Changes to the intermediate update setting takes effect on the next redraw or refresh operation. The following JavaScript functions disable and re-enable intermediate updates:

```
function disableIntermediateUpdates()
{
    var map = getMap();
    var status = map.getIntermediateUpdatesEnabled();
    if (status == true) map.setIntermediateUpdatesEnabled(false);
}

function enableIntermediateUpdates()
{
    var map = getMap();
    var status = map.getIntermediateUpdatesEnabled();
    if (status == false) map.setIntermediateUpdatesEnabled(true);
}
```

Handling Errors

Every application should track and handle errors. The Autodesk MapGuide Viewer API has error tracking methods and properties you can use to debug your applications.

Getting Error Codes

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.

To get the most recently called method error code

- 1 Call the `MGMap.getLastError` method to get the `MGError` object.
- 2 Next call the `MGError.getCode` method to get the error code. For example:

```
function checkErrorCode()
{
    var map = getMap();
    var code = map.getLastError().getCode();
    alert("ERROR: " + code);
}
```

Checking for Incorrect Argument Types

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 `MGError.getArg` method.

To check for an incorrect argument type for the most recently called method

- 1 Call the `MGMap.getLastError` method to get the `MGError` object.
- 2 Next call the `MGError.getArg` method to get the number of the incorrect argument. For example:

```
function checkArgType()
{
    var map = getMap();
    var arg = map.getLastError().getArg();
    alert("ERROR: " + arg);
}
```

To see the argument types for any API method, locate that method's Help topic in the *Autodesk MapGuide Viewer API Help*.

Debugging an Application

In addition to checking `MGError`, you can call the `MGMap.enableApiExceptions` and `MGMap.disableApiExceptions` methods 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.

Working with Map Layers, Map Features, and Printing

4

This chapter shows you how to write code for common tasks your application can perform with the Autodesk MapGuide® Viewer API. Once your application can manipulate a map, you may want it to control other Autodesk MapGuide Viewer events, as described in Chapter 5, “Handling Events.”

- Overview
- Working with map layers
- Working with map features
- Working with printing

Overview

Throughout this chapter you will find simple JavaScript code samples that show you how to perform basic Autodesk MapGuide Viewer tasks in your application. You will learn how to work with map layers, features, and how to customize map printouts.

Note that we refer to JavaScript code modules as *functions*, reserving the term *method* for the Autodesk MapGuide Viewer API. Also note that although spatial data on the map consists of *map features*, the methods and properties in the Autodesk MapGuide 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 to 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.

Working with Map Layers

This section describes common tasks your application can perform with Autodesk MapGuide map layers.

Counting Map Layers

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

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

The function starts by calling the `getMap` function and assigning its return value to a variable called `map`:

```
var map = getMap();
```

Remember that `getMap` 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 "Map Access for ActiveX Control and Plug-In" on page 32).

Next, the `countLayers` function calls the `getMapLayersEx` method, an Autodesk MapGuide Viewer API method that returns an `MGCollection` object containing all the layers defined in the map. The layer collection is assigned to the `layers` variable:

```
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 the `cnt` variable:

```
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 Map Layers

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

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

Next, the `listLayers` function uses a `for` loop to cycle through the layer collection, placing all the layer names in the single `msg` variable:

```
var msg; // empty variable to hold layer names
var i; // counter variable; used by loop
for (i = 0; i < cnt; i++) // iterate layer count times
{
    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 once for each map layer. At each iteration, the loop counter variable (`i`) is incremented and the following statements are processed:

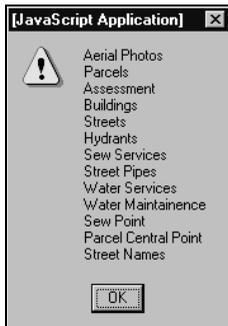
```
var layer = layers.item(i); // get next layer
msg += layer.getName() + "\n"; // add layer name to msg
```

The first statement uses the `item` 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 `getName` method 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 the variable `msg` in an alert box:

```
alert (msg);
```



`alert` box displaying layer names

Adding a Map Layer

The `doAddLayer` function adds a named layer to a map:

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

The function starts by checking the browser type. If the browser is Netscape, `doAddLayer` calls the `addMapLayer` method, supplying it with the name of an existing map layer file (MLF) and the name of the event observer:

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

If the browser is Internet Explorer, `doAddLayer` calls the `addMapLayer` method, supplying only the layer name as an argument:

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

Because `addMapLayer` takes different arguments depending on the browser type, we did not bother to return the map object with `getMap`. Instead, we supplied the map object using the syntax required by each browser.

Note If you support the Netscape browser, you must provide the name of the event observer as a second argument to `addMapLayer`.

Linking Map Layers

Using Autodesk MapGuide Author, you can 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 Autodesk MapGuide Viewer API, you can extend this functionality by linking layers to one or more designated control layers. Then if an action, such as zooming 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.

The function is named `onViewChanging` because it's triggered by the Autodesk MapGuide Viewer API event of the same name. Whenever an event is triggered, Autodesk MapGuide Viewer checks for a function whose name matches the event name. If the function is found, Autodesk MapGuide Viewer invokes it, passing arguments that vary by event.

`onViewChanging` takes an `MGMap` object as an argument passed by the `onViewChanging` event. Because the event provides an instance of the map object, we don't need to obtain it with `getMap`. For example, the following function links map layers:

```
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() ||
            countries.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` method to return each of the control layers as objects. Those objects are assigned to three variables named `states`, `counties`, and `zipCodes`.

Next, `onViewChanging` uses the `getVisibility` method to determine if any of the control layers are visible. If at least one control layer is visible (that is, if `states` is visible *or* `counties` is visible *or* `zipCodes` is visible), `getVisibility` returns the boolean value `True`, thus setting the `vis` variable to `True`. Otherwise, it sets `vis` to the `False` value.

Then `onViewChanging` uses the `getMapLayersEx` method to create a layer collection and assign it to the `layers` variable.

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 the `vis` variable.

Toggling Map Layer Visibility On and Off

The `layerToggle` function toggles the visibility of a map layer that is specified when the function is invoked:

```
function layerToggle(l_name)
{
  var map = getMap();
  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 and 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` 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 the `layer` variable:

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

Next, the function checks the value of the `layer` variable. 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 `setVisibility` method. This has the effect of checking the layer's visibility and returning the opposite of what it finds.

Working with Map Features

This section describes common tasks your application can perform with Autodesk MapGuide map features.

Getting Keys of Selected Map 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 = getMap();
    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 the `MGMap` object:

```
var map = getMap();
```

Then it uses two API methods to see if any map 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 map features are selected, an `alert` displays and the function terminates; otherwise, the selection is assigned to the `sel` variable:

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

Next, the `doGetKey` function calls the `getMapObjectsEx` method and passes its return value (a collection of the selected map features) to a variable called `objs`. Note that if you use `getMapObjectsEx` with a map layer, it returns an `MGCollection` object made up of map features of a map layer, but by using the method with the selection object, and by passing it null as a parameter, it returns the map 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 the `cntObjects` variable:

```
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 `msg` variable:

```
// variable to hold feature names
var msg = "Keys of selected features are:\n";
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 once for each object. At each iteration, the loop counter variable (`i`) is incremented and the following statements are processed:

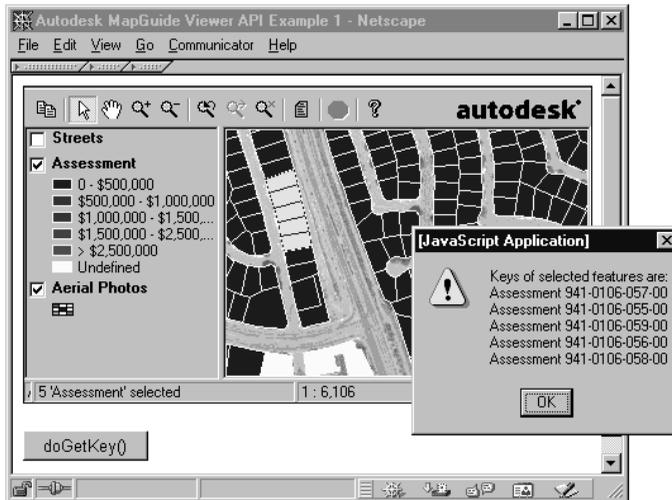
```
var obj = objs.item(i);
var key = obj.getKey();
msg = msg + obj.getMapLayer().getName() + " " + key + "\n";
```

The first statement uses the `item` 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` 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);
```

The following screen shows keys of the selected features displayed in the alert box.



alert box displaying keys of selected features

Getting Coordinates of a Selected Map Feature

The `doGetCoordinates` function displays a dialog box showing the coordinates of a selected map feature:

doGet Coordinates Function

```
function doGetCoordinates()
{
    var map = getMap();
    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");
```

doGet Coordinates Function

```
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 `doGetCoordinates` function starts by using `getMap` to get an instance of the map; then it gets the current selection and assigns it to the `sel` variable:

```
var map = getMap();
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 `Parcels` 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;
}
```

Otherwise, 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 feature. 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 `msg` variable:

```
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 once 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` 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` method:

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

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

Zooming In on Selected Features

The `zoomSelect` function zooms in to a selected feature:

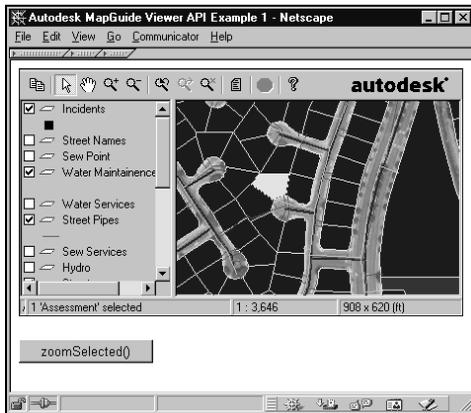
```
function zoomSelect()
{
    var map = getMap();
    var selected = map.getSelection().getMapObjectsEx(null);
    if (selected.size() > 0)
        map.zoomSelected();
    else
        alert("Nothing selected.");
}
```

First, the function gets an instance of the `MGMap` object. 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:

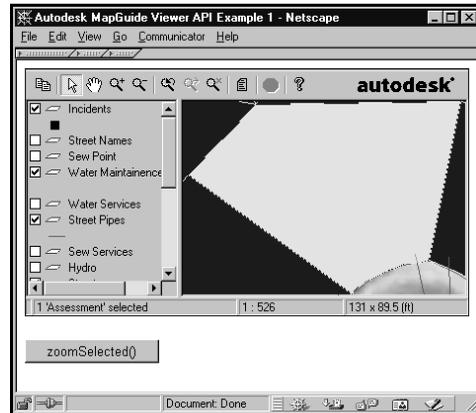
```
var map = getMap();
var selected = map.getSelection().getMapObjectsEx(null);
```

Next, `zoomSelect` uses the `size` method to see how many features are selected. If one or more features are selected, the `zoomSelected` method is invoked, causing Autodesk MapGuide Viewer to zoom to those features, as shown in the illustrations following the example. Otherwise, an alert displays:

```
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 = getMap();
    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 `showFeatureCount` 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.

Next, `showFeatureCount` creates an instance of the map and checks to see if the map is in a busy state (see “Handling Busy State and Map Refresh” on page 43 for more information about the busy state):

```
var map = getMap();
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 the `cnt` variable:

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

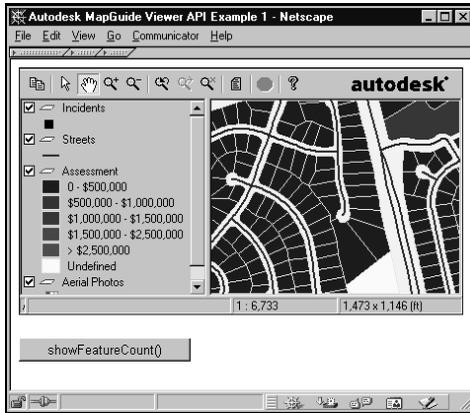
Then it creates a loop that counts the features in each layer and uses the `getLegendLabel` and `setLegendLabel` methods, as shown in the illustrations following the example, to report the map feature counts in the map legend:

```
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);
}
```

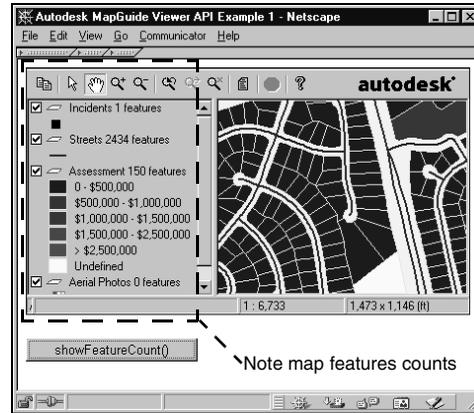
Finally `showFeatureCount` sets the global `legendSet` variable to `True`. This keeps the function from running again until the page containing the map is refreshed.

```
legendSet = true;
```

The following screens compare the results before and after calling `showFeatureCount`.



Legend before calling `showFeatureCount`



Legend after calling `showFeatureCount`

Working with Printing

Autodesk MapGuide lets map authors and Autodesk MapGuide 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. Also, when printing from Autodesk MapGuide Viewer, a 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 Autodesk MapGuide Viewer lays out the page elements that will be sent to the printer. The second event, `onEndLayout`, is called after Autodesk MapGuide 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.

Setting the Print Priority

As shown above, you can write an `onEndLayout` event handler that uses `MGPPrintLayout`, `MGPPageElement`, 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 place 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, but the default 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);
}
```

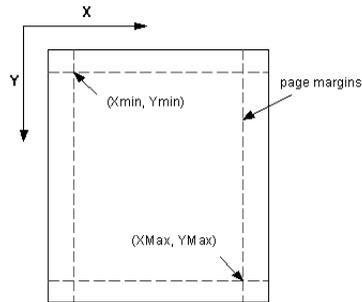
Enabling the Print Events

By default, the `onBeginLayout` and `onEndLayout` events are not fired; you enable and disable them using the `enablePrintingEvents` and `disablePrintingEvents` methods. For Autodesk MapGuide Viewer Plug-In and Autodesk MapGuide Viewer, Java Edition, you will also need to use the `setPrintingObserver` method to specify the event observer. Here is one way to write a JavaScript function that enables print events:

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

Positioning Page Elements with Page Coordinate System Units

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.



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 MWF or EMF file containing a small set of predefined symbols. However, additional symbols can be added to the list using Autodesk MapGuide® Author. Refer to the topics “Adding Symbols for Use with the Viewer API” and “API Symbol Manager dialog box” in 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);
    }
}
```

Handling Events

This chapter describes how to design your Autodesk MapGuide® Viewer application to handle Autodesk MapGuide® events. Handling events gives you more control over how your application responds to actions by users as they view maps. Once your application can handle Autodesk MapGuide Viewer events, you may want to add data query and update capabilities, as described in Chapter 6, “Using Reports to Query and Update Data Sources.”

5

- Overview
- Working with event handlers
- Setting up event handlers
- Writing event handlers

Overview

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 Autodesk MapGuide 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. 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 “SDF Component Toolkit Applications” on page 171 for an example of this). This type of function, which works only in response to an event, is called an *event handler*.

For Microsoft® Internet Explorer®, you write VBScript code to set up the event handlers for your application. VBScript, a lightweight Visual Basic-like scripting language. For Netscape® Navigator®, you need to set up *event observers*. Event observers act as the link between the event and your event handling code; they are triggered when an event occurs and then call the event handler in response.

Working with Event Handlers

Netscape Navigator and Internet Explorer handle events differently, so if you want to support both browsers, you need to write code for both. Code examples that work for both are described in “Setting Up Event Handlers” on page 72 and “Writing Event Handlers” on page 78. To see a full working event handling example, choose Help ► Contents ► Examples Advanced ► Event Handling example in the *Autodesk MapGuide Viewer API Help*.

Browser Differences

The following table summarizes the basic differences between how Netscape Navigator and Internet Explorer handle events:

Netscape Navigator	Internet Explorer
<p>For Netscape Navigator, Autodesk MapGuide provides an observer applet that you embed in your application using the <code><APPLET></code> tag. For Autodesk MapGuide Viewer Plug-In, this applet is called: <i>MapGuideObserver6.class</i></p> <p>For Autodesk MapGuide Viewer, Java Edition, it is called: <i>MapGuideObserver6J.class</i></p> <p>You can use the same instance of the observer applet for all events.</p>	<p>For Internet Explorer, you write a few lines of <code>VBScript</code> code that tell Internet Explorer the name of the event handler. You must create a separate event handler for each type of event you want to handle.</p>
<p>To call an Autodesk MapGuide method that triggers an event, you pass the Autodesk MapGuide observer as a parameter.</p>	<p>Internet Explorer knows how to find the event handler without being passed its name because it assumes the event handler name will match the event it is handling.</p>

Event Observer Set Methods

There are some `MGMap` methods whose sole function is to allow you to set an observer for a specific event. For example, when the map view changes (for example, in response to a user panning or zooming), you can set the observer for the `onViewChanged` by creating a function called `onLoad` (a browser event) and inserting the `MGMap.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.

Setting Up Event Handlers

This section describes how to set up event handling for either Autodesk MapGuide Viewer Plug-In, Autodesk MapGuide Viewer ActiveX Control, or Autodesk MapGuide Viewer, Java Edition, or for all three.

Plug-In and Java Edition Event Handlers

As described in “Browser Differences” on page 71, Autodesk MapGuide provides observer applets you can use for Autodesk MapGuide Viewer Plug-In and Autodesk MapGuide Viewer, Java Edition event handling.

- For Autodesk MapGuide Viewer Plug-In, the applet is called *MapGuideObserver6.class*.
- For Autodesk MapGuide Viewer, Java Edition, the applet is called *MapGuideObserver6J.class*.

If your application is supporting both Autodesk MapGuide Viewer Plug-In and Autodesk MapGuide Viewer, Java Edition, you need to detect which Autodesk MapGuide Viewer is present on the client’s system. You can do this by embedding a detection applet provided by Autodesk MapGuide. This applet is included in an archive file called *MGDetectClass.zip*. The detection applet can detect whether or not Autodesk MapGuide Viewer, Java Edition has been installed on a client system.

To download event observer and detection applet code, choose Help ► Contents ► Plug-In/Java Edition Downloads in the *Autodesk MapGuide Viewer API Help*. You can also download the Java source code for each applet, edit the code, and then recompile it. However, scripting with Java is highly browser-specific because of the differences in the Netscape Plug-In and Microsoft ActiveX Control embedding architectures. Therefore, we recommend that you use the observer applet as provided for event handling only.

Note If you are accessing the Autodesk MapGuide Viewer Plug-In API from a Java applet, your applet needs to function as an event observer. See “Plug-In Event Handler Example” on page 83.

To use Autodesk MapGuide detection and event observer applet

- 1 Embed the observer applet in your Web page, assigning a specific name to it with the NAME attribute of the <APPLET> tag. If you are going to use both types of applets, you must give them different names and use the correct name in your method calls after you have detected which Autodesk MapGuide Viewer your client is running.
- 2 Use the detection applet to determine which Autodesk MapGuide Viewer (Autodesk MapGuide Viewer Plug-In or Autodesk MapGuide Viewer, Java Edition) is running on the client machine.
- 3 Once you have detected which Autodesk MapGuide Viewer is running on a client machine, invoke the event observer applet by calling a set observer method like setSelectionChangedObserver or a method that invokes an event, such as digitizePolylineEx.

For example, the following code shows how to check the Autodesk MapGuide Viewer version and install the appropriate event observer applet for Autodesk MapGuide Viewer Plug-In or Autodesk MapGuide Viewer, Java Edition:

Installing Plug-In and Java Editions Observer Applets

```
<SCRIPT LANGUAGE="JavaScript">
// Embed the detect applet to check if the
// Autodesk MapGuide Viewer, Java Edition is installed
document.write('<APPLET');
document.write(' CODEBASE="detect_class"');
document.write(' ARCHIVE="MGDetectClass.zip"');
document.write(' CODE=MGDetectClass');

// Extract result from detector
tempurl = document.URL;
index = tempurl.indexOf("DETECTED=");
result = tempurl.substring(index, tempurl.length);

if (result == "DETECTED=true")
{
    // The Autodesk MapGuide Viewer, Java Edition was installed,
    // so we embed the Autodesk MapGuide Viewer, Java Edition
    // Observer Applet and name it obsJava
    document.write("<Applet CODE=\"MapGuideObserver6J.class\"
        WIDTH=2 HEIGHT=2 NAME=\"obsJava\" MAYSCRIPT>");
    document.write("</Applet>");

    // After the page loads, the browser automatically calls
    // the onLoad function.
    // onLoad calls the setSelectionChangedObserver method
    // from MGMap, providing the
```

Installing Plug-In and Java Editions Observer Applets (*continued*)

```
// Autodesk MapGuide Viewer, Java Edition
// with the observer object that handles selection
// changed events.
function onLoad()
{
    if (navigator.appName() == "Netscape")
        getMap().setSelectionChangedObserver(document.obsJava);
}

function onSelectionChanged(map)
{
    alert("Selection Changed");
}
}

// Autodesk MapGuide Viewer, Java Edition was not
// installed so we check to see if the browser is Netscape
else if (navigator.appName() == "Netscape")
{
    // The browser is Netscape, so we embed the
    // Autodesk MapGuide Viewer Plug-In Observer Applet
    document.write("<Java Applet CODE=\"MapGuideObserver6.class\"
        WIDTH=2 HEIGHT=2 NAME=\"obs\" MAYSCRIPT>");
    document.write("</Java Applet>");

    function onLoad()
    {
        if (navigator.appName() == "Netscape")
            getMap().setSelectionChangedObserver(document.obs);
    }

    function onSelectionChanged(map)
    {
        alert("Selection Changed");
    }
}
</SCRIPT>
```

ActiveX Control Event Handlers

You can handle events from the Autodesk MapGuide Viewer ActiveX Control within Microsoft Internet Explorer by defining VBScript functions.

Define VBScript functions in the HTML file where the <OBJECT> tag resides. Define a function for each of the events that you want to handle. Function names must begin with the name of the MGMap object, followed by an underscore (_), followed by the event name and parameter list. For example:

```
<SCRIPT LANGUAGE="VBScript">
Sub map_onSelectionChanged(map)
    onSelectionChanged map
End Sub
</SCRIPT>
```

When an event occurs, Internet Explorer looks for a VBScript function with the name of the object in which the event occurred followed by the event. For example, in response to an onSelectionChanged event, Internet Explorer looks for a function named mapName_onSelectionChanged. If it finds such a function, it treats it as the event handler.

Plug-In and ActiveX Control Event Handlers

The following examples support event handling for both Netscape Navigator and Internet Explorer.

VBScript Example

This VBScript example is designed to handle the onDigitizedPoint event for both Netscape Navigator and Internet Explorer:

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

// Netscape Navigator Observer Applet: MapGuideObserver6.class
// ...if Netscape, embed event observer
if (navigator.appName == "Navigator")
{
    document.write("<APPLET CODE=\"MapGuideObserver6.class\"
WIDTH=2 HEIGHT=2 NAME=\"obs\" MAYSCRIPT>");
    document.write("</APPLET>");
}
}
```

We named the event observer `obs`. To call this observer for Netscape Navigator, refer to the observer as `document.obs`, because in the Netscape object model the observer is an object of the document. Be sure to copy the Autodesk MapGuide Viewer Plug-In observer applet code `MapGuideObserver6.class` to the same directory as your HTML files, or this code won't work. If you were to embed `MapGuideObserver6.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. For example: `parent.mapframe.document.obs`.

Now, suppose you have a button named `Digitize`. This button is set up so that its `onClick` event (a browser event) calls a function you created, `DigitizeIt`. The `DigitizeIt` function calls the `digitizePoint` method, an Autodesk MapGuide Viewer API method that waits for the user to click a point on the map and then captures that point. For Netscape Navigator, the `digitizePoint` method requires an event observer as a parameter, whereas Internet Explorer does not. Here is the 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. 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.

JavaScript Example

The following JavaScript example shows an HTML page that handles the `onSelectionChanged` event for both Netscape Navigator and Microsoft Internet Explorer browsers:

```
<SCRIPT LANGUAGE="JavaScript">
  if (navigator.appName() == "Netscape")
  {
    // Embed the Autodesk MapGuide Viewer Plug-In Observer Applet
    document.write("<Java Applet CODE=\"MapGuideObserver6.class\" "
WIDTH=2 HEIGHT=2 NAME=\"obs\" MAYSCRIPT>");
    document.write("</Java Applet>");
  }

  function onLoad()
  {
    if (navigator.appName() == "Netscape")
      getMap().setSelectionChangedObserver(document.obs);
  }

  function onSelectionChanged(map)
  {
    alert("Selection Changed");
  }
</SCRIPT>
```

If this script is loaded in Netscape Navigator, it first adds an `<APPLET>` tag to the document, which loads the *MapGuideObserver6.class* applet. The `<APPLET>` tag must be added in this manner or the page will not load correctly in Internet Explorer. The applet is given the name `obs`. After the page loads, the browser automatically calls the `onLoad` function. `onLoad` calls the `setSelectionChangedObserver` method from `MGMap`, providing the Autodesk MapGuide Viewer Plug-In and its observer applet with the observer object that handles selection changed events. Finally, the `map_onSelectionChanged` function is implemented to handle the event in JavaScript. This one function will now handle selection changed events from both the Autodesk MapGuide Viewer Plug-In and Autodesk MapGuide Viewer ActiveX Control.

To see a fully functional event handling example, choose [Help ► Contents ► Examples Advanced ► Event Handling in the Autodesk MapGuide Viewer API Help](#). This example implements an observer object that supports all of the observer interfaces by forwarding the events back to JavaScript functions that have identical semantics to the Autodesk MapGuide Viewer ActiveX Control event handlers.

VBScript and JavaScript Example

The following VBScript and JavaScript example shows an HTML page that handles the `onSelectionChanged` event for both Netscape and Microsoft browsers. This example forwards the event from your VBScript function to a shared JavaScript function that processes the event. For example, the following function handles the `onSelectionChanged` event for an `MGMap` object called `map`:

```
// This VBScript function handles events from the
// Autodesk MapGuide Viewer ActiveX Control
// control and passes it to the JavaScript function defined
// below. Netscape browsers will ignore the VBScript methods.

<SCRIPT LANGUAGE="VBScript">
Sub map_onSelectionChanged(map)
    onSelectionChanged map
End Sub
</SCRIPT>

// The following JavaScript function processes events
// for both the Autodesk MapGuide Viewer ActiveX Control and
// Autodesk MapGuide Viewer Plug-In.
// Autodesk MapGuide Viewer ActiveX Control events
// are forwarded to this function via VBScript, and
// Autodesk MapGuide Viewer Plug-In events are
// forwarded to this function via the
// Autodesk MapGuide Viewer Plug-In observer.

<SCRIPT LANGUAGE="JavaScript">
function onSelectionChanged(map)
{
    alert("Selection Changed");
}
</SCRIPT>
```

Writing Event Handlers

To make an Autodesk MapGuide observer work properly, always give your event handler the same name as the event it is handling. When the event is fired, the code in the corresponding observer is executed automatically. This section describes how to write JavaScript and Java event handlers for the `onBeginLayout` and `onEndLayout` events.

Page Setup Event Handler Example

If you want your application to control settings from the Autodesk MapGuide Viewer Page Setup dialog box, you can write event handling code that responds to the `onBeginLayout` event. When `onBeginLayout` is triggered, it automatically passes two objects, the `MGPageSetup` and `MGPrintInfo` objects:

```
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 in JavaScript that suppresses all page elements except the map. The example assumes you've set up the event handlers as described in "Setting Up Event Handlers" on page 72, and enabled print events as described in "Enabling the Print Events" on page 66.

To suppress page elements except the map

- 1 Create a button on the HTML page:

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

- 2 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";
  getMap().printDlg();
  map_only = "false";
}
```

- 3 Write the event handler. It goes in the HTML page (or the JavaScript `.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` object and an `MGPrintInfo` object 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);
    }
}
```

The following code shows how to implement the same event handler for the `onBeginLayout` event in Java:

```
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);
    }
}
```

Note You can control the Page Setup without using the `onBeginLayout` event, 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.

Print Event Handler Example

If you want your application to change the title font, you can add a custom symbol, or control the position and size of any page element. To do this, you need to write event handling code that responds to the `onEndLayout` event. When `onEndLayout` is triggered, it automatically passes two objects, `MGPriLayout` and `MGPriInfo` objects:

```
void onEndLayout (MGPriLayout prLayout, MGPriInfo info)
```

The `MGPriLayout` object provides access to printed page elements. You can then use `MGPriPageElement` and `MGPriExtentEx` to control how those elements display. The `MGPriInfo` 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 a print event handler in JavaScript that adds a custom symbol (`myLogo`) to the printout. The example assumes you've set up the event handlers as described in "Setting Up Event Handlers" on page 72, and enabled print events as described in "Enabling the Print Events" on page 66.

To add a custom symbol to the printout

- 1 Create a button on the HTML page:

```
<form>
  <input type="button" value="Add Symbol"
        onClick="add_symbol();" name="myButton">
</form>
```

- 2 Create a JavaScript function that the button will call. In this example, the function sets the state of a boolean variable called `new_symbol`. The variable will be read by our event handler, so we've given it global scope by declaring it outside the function body:

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

- 3 Write the event handler. It goes in the HTML page (or the JavaScript `.js` file), just like any other JavaScript function. This function is executed automatically every time the `onEndLayout` event fires. Note that the function takes an `MGPrintLayout` object and an `MGPrintInfo` object as its parameters:

```
function onEndLayout (layout, info)
{
    if (new_symbol == "true")
    {
        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);
    }
}
```

The following code shows how to implement the same event handler for the `onEndLayout` event in Java:

```
public class MyObserver extends Applet implements
MGPrintingObserver
{
    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);
    }
}
```

Plug-In Event Handler Example

To handle events in an applet for the Autodesk MapGuide Viewer Plug-In, you need to implement the corresponding Autodesk MapGuide Viewer event handler interfaces in your applet.

To implement event handler interfaces

- 1 Implement the interface of the event that you want to handle in your applet.

For instance, if you want to handle the `onDigitizedCircle` event, you need to implement the `MGDigitizeCircleObserver` interface in your applet. The `MGDigitizeCircleObserver` observer contains one method, `onDigitizedCircle`. To implement `MGDigitizeCircleObserver`, your applet must implement this method. For example:

```
public class myApplet extends applet implements
MGDigitizeCircleObserver
{
    public void onDigitizedCircle(MGMap map, String units,
        MGPoint center, double radius)
    {
        // Place your event-handling code here
    }
    ...
}
```

Now your applet can act as an event observer for the `onDigitizedCircle` event.

- 2 Every time you call the method that triggers the event, pass in the name of your applet as the second parameter.

For example, whenever you call `digitizeCircle` method in your applet, you must pass in your applet name (in this case, the keyword `this` since the applet is referring to itself) as the second parameter. For example:

```
Public void testDigitizedCircle()
{
    myUSMap.digitizeCircle("", this);
}
```

When Autodesk MapGuide Viewer Plug-In triggers the `onDigitizedCircle` event, the applet will execute the code you implemented for `onDigitizedCircle` in your applet.

Using Reports to Query and Update Data Sources

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, Macromedia® ColdFusion® and Microsoft® Active Server Pages.

6

- Overview
- Introducing ColdFusion and ASP
- Creating report scripts with ColdFusion
- Creating report scripts with ASP

Overview

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 can 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

The role of Autodesk MapGuide 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 you want 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 code might look like this:

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

This URL code 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 in Autodesk MapGuide Author.

Launching the Report

You can launch the View Reports dialog box from Autodesk MapGuide Viewer by right-clicking the map and choosing View ► Reports, or by clicking 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`.

Note Autodesk MapGuide Viewer ActiveX Control cannot open reports whose names contain double quotation marks, such as "Server Report". Therefore, if you are developing for Autodesk MapGuide Viewer ActiveX Control, do not use double quotation marks in your report names. Autodesk MapGuide Viewer ActiveX Control can open reports whose names contain single quotation marks, such as 'Server Report'.

Introducing ColdFusion and ASP

The examples in this chapter were created using two report engines, Macromedia 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.

This developer's guide 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. ColdFusion is available from Macromedia, and ASP is included as part of Windows 2000 and Windows NT Server 4.0 with the Windows NT 4.0 Option Pack. Although the examples are specific to ColdFusion and ASP, the concepts are general, applying to Common Gateway Interface (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 pages you produce can be read by any Web *browser*. The limitation exists only for the Web *server*.

Creating Report Scripts with ColdFusion

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 109.

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 the Autodesk MapGuide 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 the 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>
```

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.

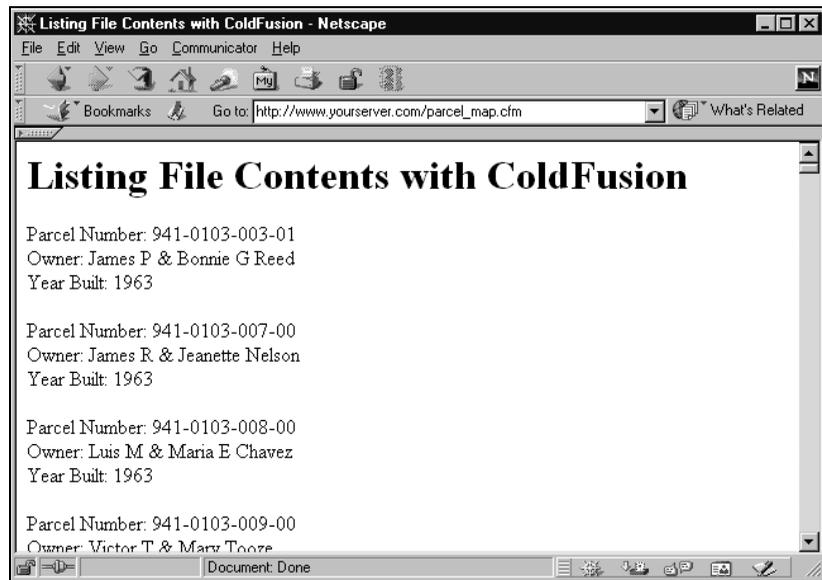
However, because this particular table has more than 5,000 records, selecting everything in it might not be such a good idea. Let's limit the output by showing only houses built in 1963. To do so, go back to `<CFQUERY>` and change the SQL statement to the following:

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

Here's a listing of the complete CFM file called *parcel_report.cfm*:

```
<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 following shows the page as it appears in a browser.



The HTML output

In this example, the database happens to be an Autodesk MapGuide resource, but it could be anything: a Microsoft® Access database listing employees and their phone numbers, a Microsoft® 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.

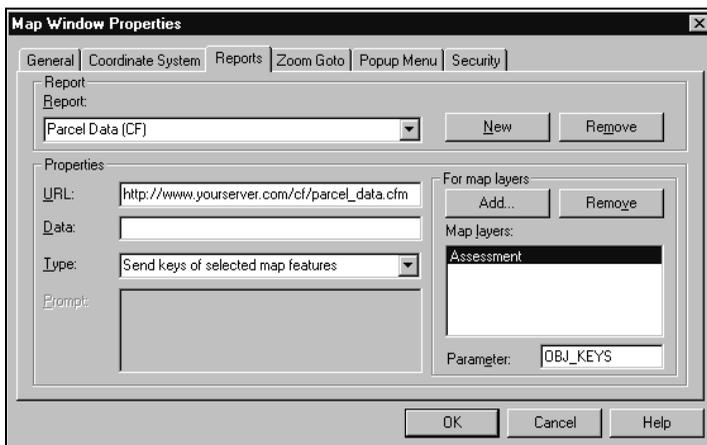
Querying and Displaying Data via the Map with ColdFusion

Now that we've seen how ColdFusion works, let's use it with Autodesk MapGuide. This example uses the *StarterApp.mwf* file 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 shows 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 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 could 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 could 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 code 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 with ColdFusion,” on page 89). 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 SQL statement now points to a ColdFusion variable, `#PreserveSingleQuotes(OBJ_KEYS)#`. The `OBJ_KEYS` values refer 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.

The SQL statement is basically saying in `Parcel_Data` to select all records whose `APN` field matches `OBJ_KEYS`. In other words, 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

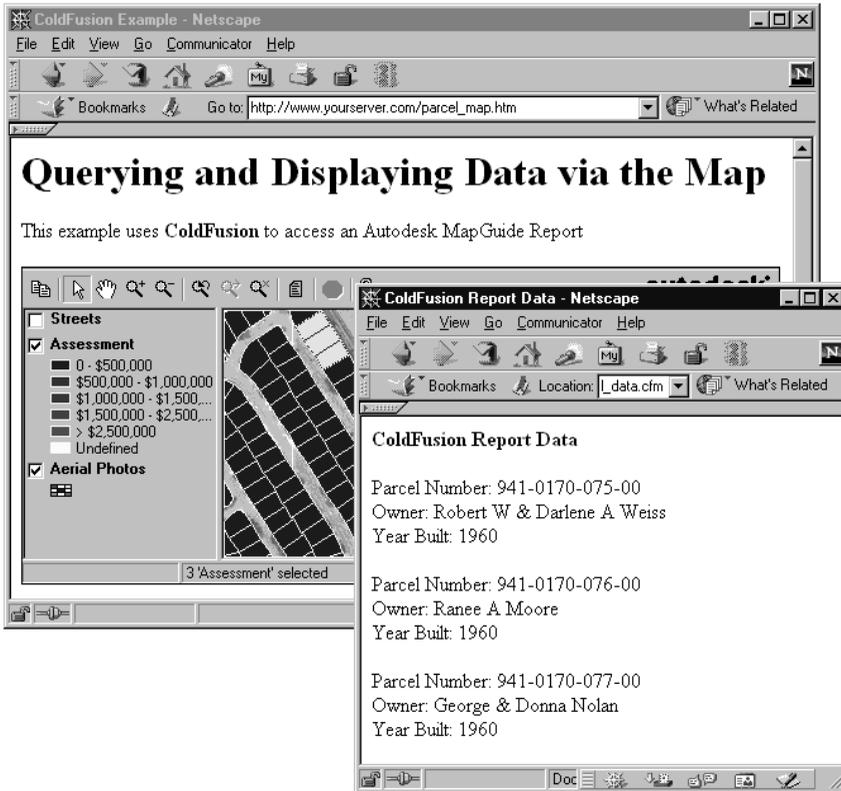
The last step is to create an HTML page to display our map. The following is for the *parcel_map.htm* file:

```
<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="map" 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="map" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Seeing the Results

Users can now generate a report for one or more map features by selecting the map features, right-clicking and choosing View ► Reports from the popup menu, and then selecting Parcel Data (CF).

The following illustration shows *parcel_map.htm* in the Web browser.



Displaying the parcel data report in a new window

Redirecting Report Output

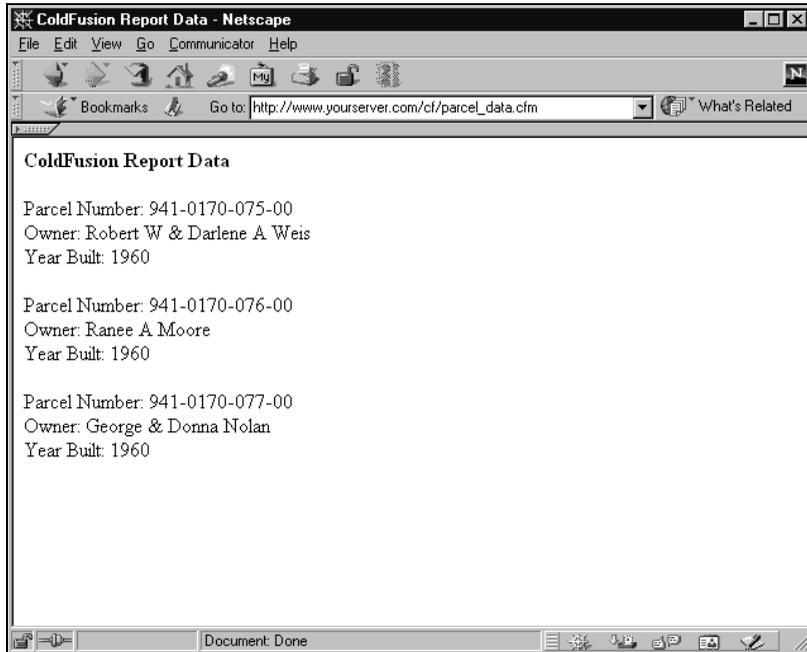
To avoid cluttering the desktop, let's generate the report in the current browser window, instead of displaying it in a new instance of the browser. Go back to the *parcel_map.htm* file and modify the embedded map code:

```
<OBJECT ID="map" 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="map"WIDTH=600 HEIGHT=250>
</OBJECT>
```

Notice that we've added a Autodesk MapGuide 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 parcel data 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.

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:

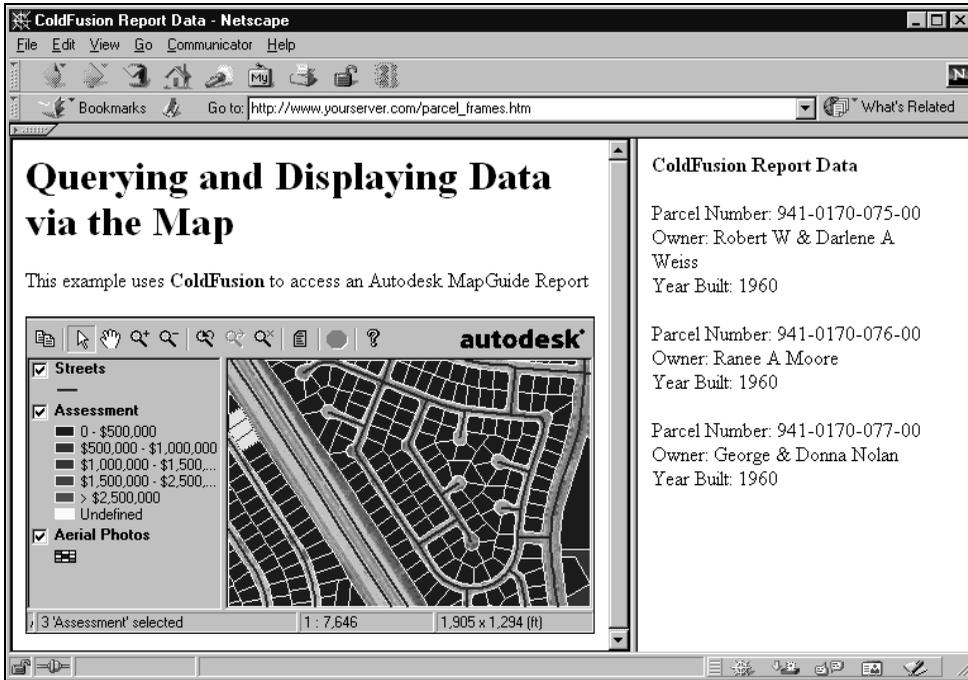
```
<HTML>
<HEAD>
<TITLE>ColdFusion Report Data</TITLE>
</HEAD>
<!-- frames -->
<FRAMESET COLS="65%,*">
<FRAME NAME="Left" SRC="parcel_map.htm" MARGINWIDTH="10"
MARGINHEIGHT="10" SCROLLING="auto" FRAMEBORDER="yes">
<FRAME NAME="Right" SRC="about:blank" MARGINWIDTH="10"
MARGINHEIGHT="10" SCROLLING="auto"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

Notice that we have 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 us go back to the `parcel_map.htm` file and change the `ReportTarget` parameter to `Right`, the name we assigned to our right-hand frame:

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

The following illustration shows the map and the report in two frames of the same window.



Displaying the map and report in frames

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

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 by creating a Parcel Report button that will display the report.

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

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

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

The `<SCRIPT>` tag holds two JavaScript functions. The first function is `getMap`. The second, `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. You will notice that the text instructs users to select one or more map features. This instruction displays each 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 the `viewReport` method to run our report. The statement begins by calling `getMap`, which returns the map object in the Web page. The map object is then passed to `viewReport`, which directs Autodesk MapGuide to display Parcel Data (CF).

Now that our function is defined, we need a way to call it, adding 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.

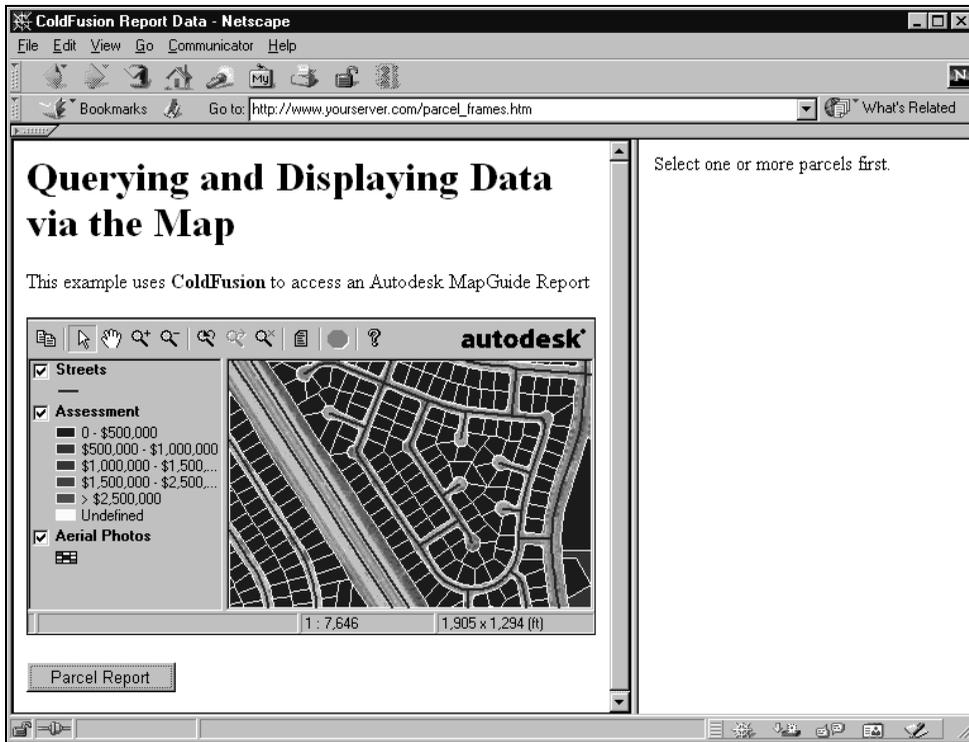
Note A JavaScript function must appear above the JavaScript code that calls it. This keeps users from trying to call a function before it has been parsed by the browser. JavaScript functions are typically defined in a single <SCRIPT> tag in the <HEAD> section of the HTML file.

The following code is the finished application:

```
<HTML>
<HEAD>
<TITLE>ColdFusion Example</TITLE>
<!-- JavaScript functions -->
<SCRIPT>
// function #1
function getMap()
{
    if (navigator.appName == "Netscape")
        return parent.Left.document.map;
    else
        return parent.Left.window.map;
}
// function #2
function runReport()
{
    parent.Right.document.write("<P>Select one or more parcels first.</P>");
    getMap().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="map" WIDTH=600 HEIGHT=250
CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
<PARAM NAME="URL"
VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=
Right">
<EMBED
SRC="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=Ri
ght NAME="map" WIDTH=600 HEIGHT=250>
</OBJECT>
<!-- Parcel Report button -->
<FORM>
<INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()">
</FORM>
</BODY>
</HTML>
```

The following shows the application results if no map features have been selected.



Displaying the map, a report frame, and a Parcel Report button

Modifying a Database via the Map with ColdFusion

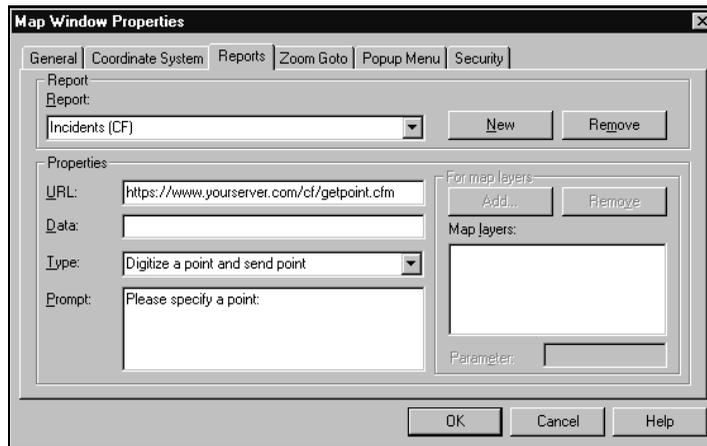
This example shows how to create an application that lets users add points to a map from their browsers. The added points are then stored in a database on the server and are visible to anyone else viewing the map. The example shows a hypothetical Incident Log application that will be used to track crimes 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 the *getpoint.cfm* file.

- 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 that prompts 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 with coordinate values of 37.721,-121.943, the URL would be:

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

Creating the Report Scripts

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

The first CFM 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.

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`. 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=" + "37.721" + "&LON=" + "-121.943";
```

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

The next line creates a new browser window, using the `loc` variable to supply the URL. The last line shifts browser focus to the new window we just created.

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

```
<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>
```

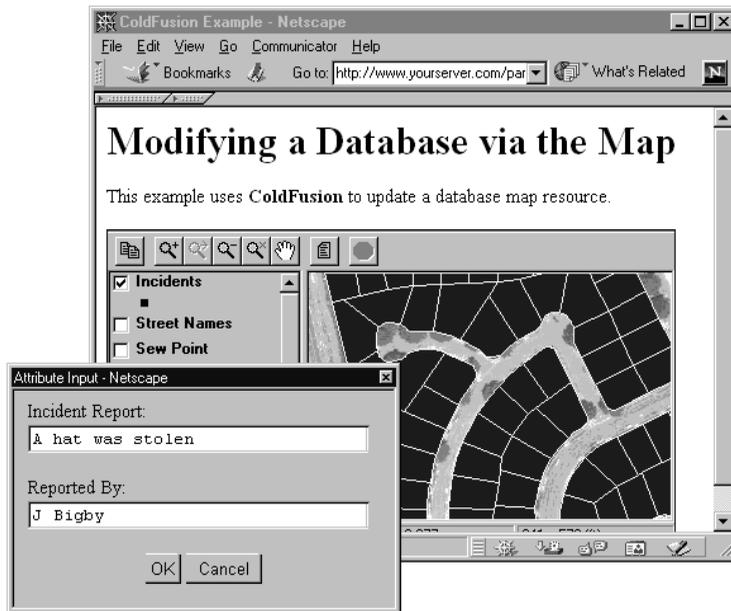
The *showform.cfm* file does indeed show a form, which is used to enter a description of the incident.

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.

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*.



Data entered in the *showform.cfm* form

Specifying the lat/lon point 37.721,-121.943 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 the *insert.cfm* file:

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

Now, let's see how the *insert.cfm* file 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 Insert statement, which is used to add the form data to the map resource database.

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 the URL parameters passed by *showform.cfm*.

The first line of the `<SCRIPT>` element 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

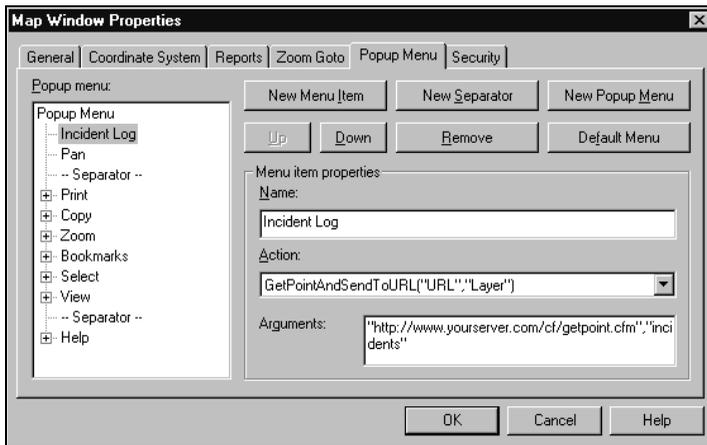
To create an HTML page to display our map, we'll use *parcel_map.htm*, modifying only the `<H1>` and the short paragraph of descriptive text:

```
<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="map" WIDTH=600 HEIGHT=250
CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
<PARAM NAME="URL"
VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=
Right">
<EMBED
SRC="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=Ri
ght NAME="map" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Creating a Custom Menu Item

There are a few problems with this design. One is that it requires users to select View ► Reports from the popup menu, then select Incidents (CF) from the list, then clear the JavaScript alert box that tells them to select a point, and then digitize the point. Another 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 options, as shown in the following illustration, on the Popup Menu tab of the Map Window Properties dialog box.



Dialog box specifications for Incident Log popup 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 Autodesk MapGuide 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 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

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 the need for the user to reload the map manually, you can use the Autodesk MapGuide 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

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 (extension `.asp`). Instead of tags, ASP files include scripts written in VBScript, a lightweight Visual Basic-like scripting language, or in JScript, the Microsoft version of JavaScript.

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 ASP functionality is provided by *objects* and *components*. Objects and components are ActiveX Control 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.asp` 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

The following tables summarize and describe objects, components, and events you use to create a report script with ASP.

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

Objects Used to Create a Report Script

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 <code>CreateObject()</code> , which you'll use to create a connection to your database map resources.

Components Used to Create a Report Script

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.

Events Used to Create a Report Script

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 89.

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 the Autodesk MapGuide reporting feature. Later, we'll see how Autodesk MapGuide fits into the picture.

Let's say you have an MWF file pointing to a database containing parcel information, such as the lot number, street address, owner's name and so on. If you want to list the contents of the database at the bottom of an HTML page displaying the map, you would first rename the HTML file with an *.asp* extension and place it in a directory with script or execute permissions. Then you would 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, the Microsoft 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 the server-side code that ASP should process.

Selecting Database Records

Next, we'll add 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 `Assessor`, 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 the `dbConnection` variable.

```
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 the `dbConnection` variable.

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

The third line creates a variable that holds a SQL statement specifying the database records for 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 accustomed 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 another that finishes up the loop structure:

```
RS.MoveNext  
Loop
```

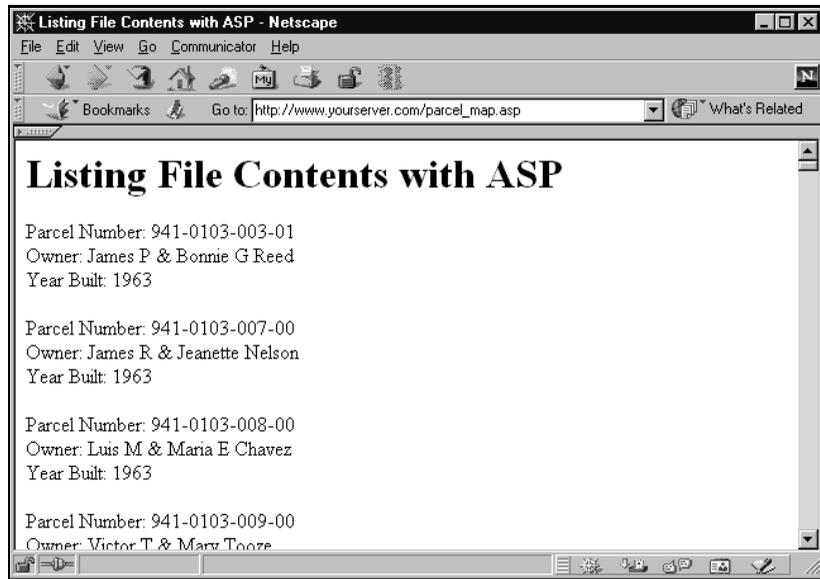
Now we're ready to load the page in our browser.

Seeing the Results

Here's a listing of the complete ASP file *parcel_report.asp*:

```
<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 following is the page as it appears in a browser.



The HTML output

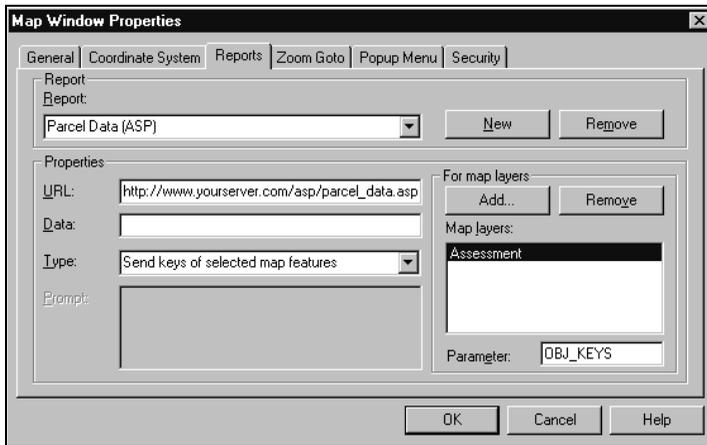
Like the earlier ColdFusion example, this page is very simple, only hinting at the power of what you can do with ASP. And like the earlier example, this is not really an Autodesk MapGuide application. The database happens to be an Autodesk MapGuide resource, but it could be any database you have access 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.

Querying and Displaying Data via the Map with ASP

Now that we've seen how ASP works, let's use it with Autodesk MapGuide. This example uses the *StarterApp.mwf* file 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 (ASP) and will access a server page whose URL is http://www.yourserver.com/asp/parcel_report.asp. The server page will display information about user-selected features on a layer called Assessment. The Map Window Properties dialog box shows 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 could 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 code 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 ("Listing File Contents with ASP" on page 111). The only change is to the value we assign the SQLQuery variable.

As with the previous example, the statement is selecting records from the `Parcel_Data` DSN. The difference is that the SQL statement 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 to select in `Parcel_Data` all records whose APN field matches `OBJ_KEYS`. In other words, 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.

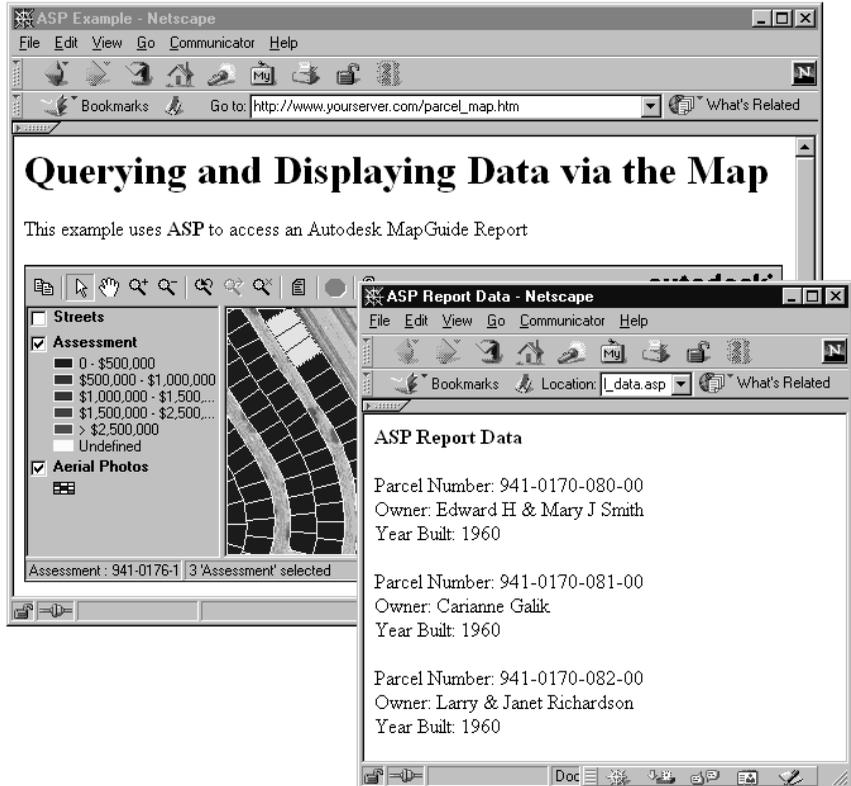
Creating an HTML Page to Display the Map

The last step is to create an HTML page to display our map. The following listing is for the `parcel_map.htm` file:

```
<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
Report</P>
<!-- embedded map -->
<OBJECT ID="map" 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="map"
WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Seeing the Results

We're ready to view *parcel_map.htm* in our Web browser, as shown in the following illustration. Now, users can generate a report on one or more map features by selecting the features, right-clicking and selecting View ► Reports from the popup menu, and then selecting Parcel Data (ASP).



Displaying the parcel data report in a new window

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

Redirecting Report Output

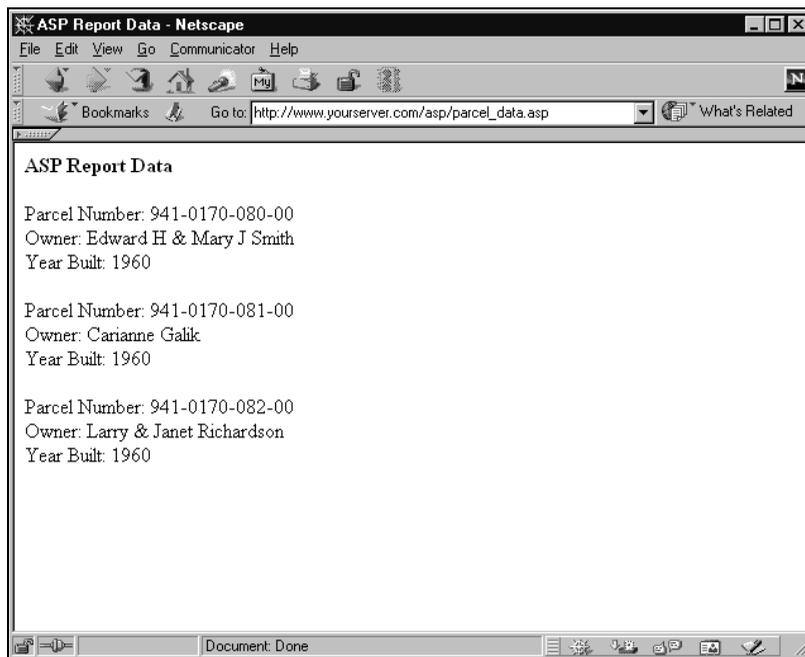
To avoid cluttering the desktop, let's generate the report in the current browser window, instead of displaying it in a new instance of the browser. Go back to the *parcel_map.htm* file and modify the embedded map code:

```
<OBJECT ID="map" 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="map"WIDTH=600 HEIGHT=250>
</OBJECT>
```

Notice that we've added a Autodesk MapGuide 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 parcel data 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>
```

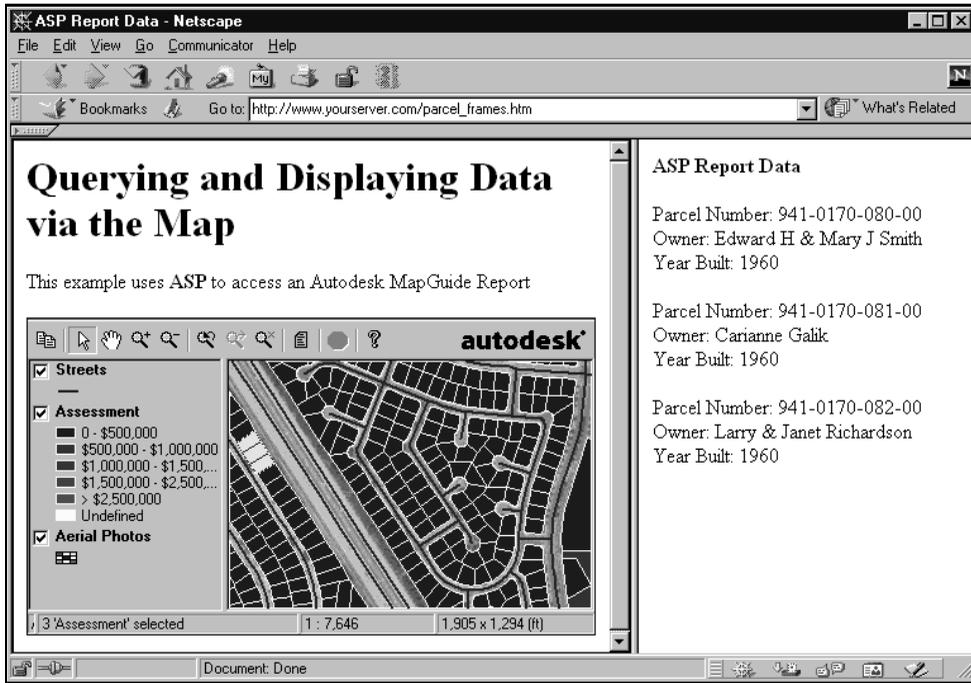
Notice that we've assigned the names `Left` and `Right` to the frames. The source for `Left` is `parcel_map.htm`, the file containing our embedded map. The source for `Right` is `about:blank`, a standard browser function whose purpose is to display a blank window or frame.

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

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

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.

The following illustration shows with the map and the report in two frames of the same window.



Displaying the map and report in frames

Adding a Button with the Viewer API

An Autodesk MapGuide report is generated by right-clicking the map and then choosing View ► Reports from the popup menu. We'll create a Parcel Report button that will display the report.

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

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

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

The `<SCRIPT>` tag holds two JavaScript functions: `getMap` and `runReport`, which 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, instructing users to select one or more map features. This instruction displays each 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 the `viewReport` method to run our report and begins by calling `getMap`, which returns the appropriate map feature. That feature is then passed to `viewReport`, which directs Autodesk MapGuide to display Parcel Data (ASP).

Now that our function is defined, to call it we'll add a `<FORM>` element to `parcel_map.htm`:

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

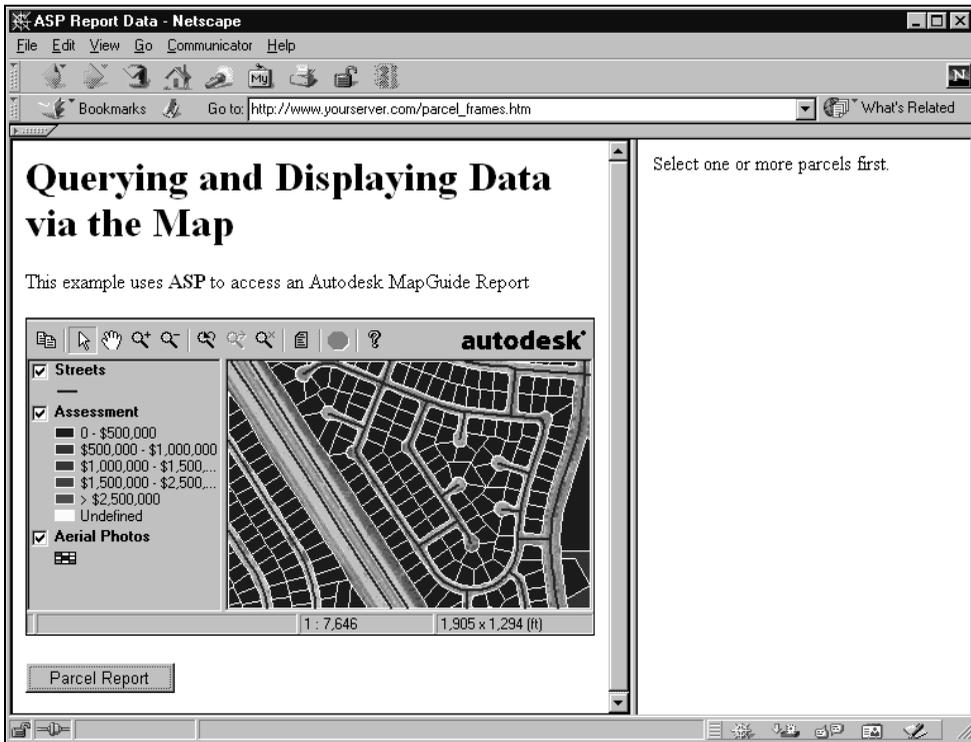
Note A JavaScript function must appear above the JavaScript code that calls it. This keeps users from trying to call a function before it has been parsed by the browser. JavaScript functions are typically defined in a single `<SCRIPT>` tag in the `<HEAD>` section of the HTML file.

This the final text of *parcel_map.htm*, as well as the finished application:

parcel_map.htm

```
<HTML>
<HEAD>
<TITLE>ASP Example</TITLE>
<!-- JavaScript functions -->
<SCRIPT>
// function #1
function getMap()
{
    if (navigator.appName == "Netscape")
        return parent.Left.document.map;
    else
        return parent.Left.window.map;
}
// function #2
function runReport()
{
    parent.Right.document.write("<P>Select one or more parcels first.</P>");
    getMap().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
Report</P>
<!-- embedded map -->
<OBJECT ID="map" WIDTH=600 HEIGHT=250
CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
<PARAM NAME="URL"
VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=
Right">
<EMBED
SRC="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=Ri
ghtNAME="map" WIDTH=600 HEIGHT=250>
</OBJECT>
<!-- Parcel Report button -->
<FORM>
<INPUT TYPE="button" VALUE="Parcel Report" ONCLICK="runReport()" </INPUT>
</FORM>
</BODY>
</HTML>
```

The following illustration shows the results of the application if no map features have been selected.



Displaying the map, a report frame, and a Parcel Report button

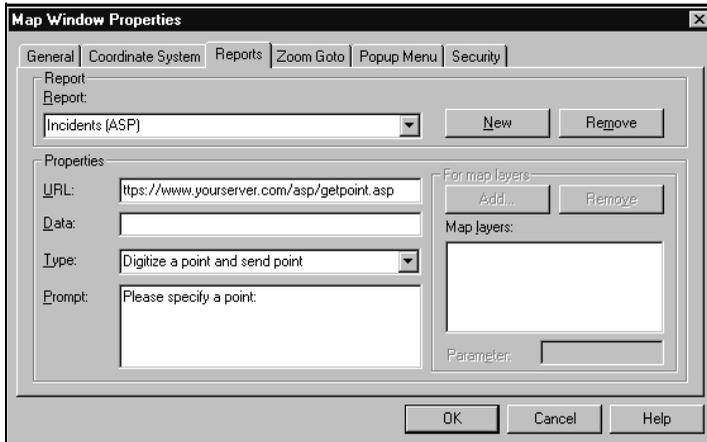
Modifying a Database via the Map with ASP

Users can add points to a map from their browsers. The added points are then stored in a database on the server and are visible to anyone viewing the map. The example shows 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 the *getpoint.asp* file.
- 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

The following show 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 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 with coordinate values of 37.721,-121.943, the URL would be:

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

Creating the Report Scripts

Next we'll create the three ASP 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:

```
<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 in the previous example. The first line of code closes that window.

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`. 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=" + "37.721" + "&LON=" + "-121.943";
```

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

The next line creates a new browser window, using the `loc` variable to supply the URL. The last line shifts browser focus to the new window we just created.

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

```
<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 shows a form, used to enter a description of the incident. The form follows HTML syntax, but also contains the ASP variables:

```
<%=Request.QueryString("lat")%>
<%=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 added to the URL directly, not generated by a form

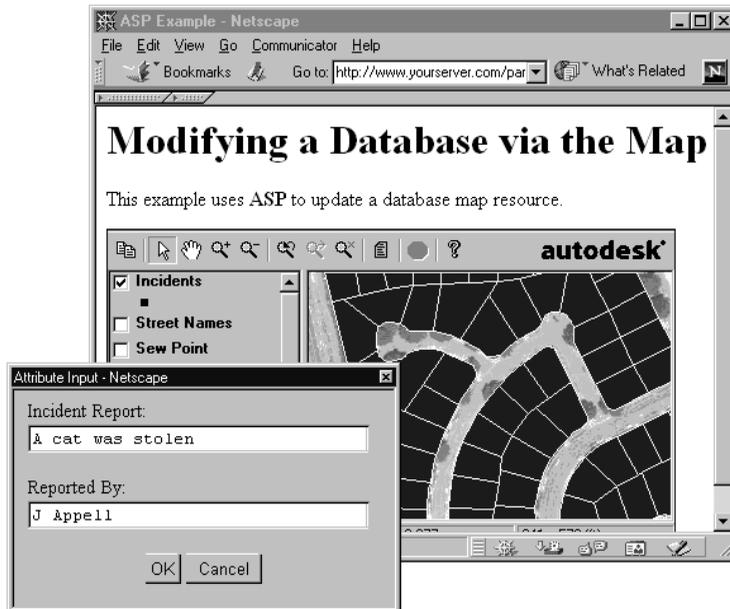
We use `QueryString` in this case, because the URL parameters that were sent to *showform.asp* were created explicitly by JavaScript code in *getpoint.asp*.

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.

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*.



Data entered in the *showform.asp* form

Specifying the lat/lon point 37.721,-121.943 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 the *insert.asp* file:

```
insert.asp?rpt_lat=37.721&rpt_lon=-121.943&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.

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`.

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`.

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

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('37.721','-121.943','A cat was stolen','J Appell')"
```

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

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. To avoid this, add code to replace a single apostrophe with two apostrophes as follows: Change `Request.Form("rpt_info")` to `Replace(Request.Form("rpt_info"), "'", "'')`. See your ASP documentation for more information.

In the `<SCRIPT>` element, 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

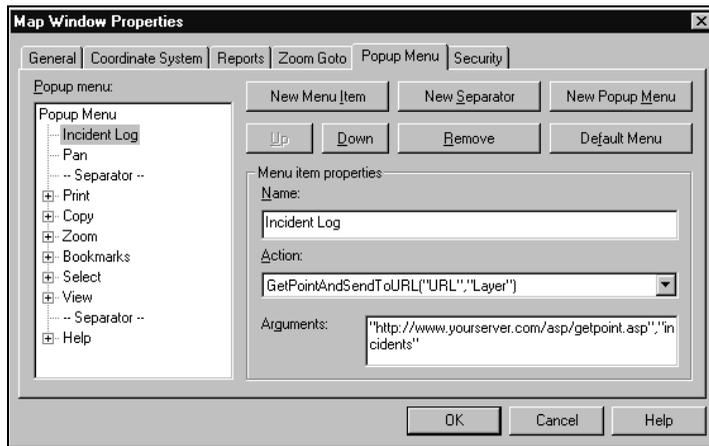
The last step is to create an HTML page to display our map. We'll use the *parcel_map.htm* file from the previous example, modifying the `<H1>` and the short paragraph of descriptive text, but leaving the rest of the file unchanged:

```
<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="map" WIDTH=600 HEIGHT=250
CLASSID="CLSID:62789780-B744-11D0-986B-00609731A21D">
<PARAM NAME="URL"
VALUE="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=
Right">
<EMBED
SRC="http://www.yourserver.com/maps/StarterApp.mwf?ReportTarget=Ri
ghtNAME="map" WIDTH=600 HEIGHT=250>
</OBJECT>
</BODY>
</HTML>
```

Creating a Custom Menu Item

There are a few problems with this design. One is that it requires users to select View ► Reports from the map-window popup menu, then select Incidents (ASP) from the list, then clear the JavaScript `alert` box that tells them to select a point, and then digitize the point. Another 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. Select the options, as shown in the following illustration, on the Popup Menu tab of the Map Window Properties dialog box.



Dialog box specifications for Incident Log popup 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 Autodesk MapGuide 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

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 need for the user to reload the map manually, you can use the Autodesk MapGuide 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.

Applications

7

This chapter demonstrates various features of the Autodesk MapGuide® Viewer API by describing and including source code for several applications.

- Overview
- Custom redlining application
- Municipal application
- Facility management application
- DWG filtering application
- SDF Component Toolkit applications

Overview

This chapter provides examples of how you might use the Autodesk MapGuide Viewer API to create real-world applications. Be sure to also visit the Autodesk MapGuide Web site at www.autodesk.com/mapguide for additional information about Autodesk MapGuide development, including demo applications, real customer sites, and other developer resources.

When you have finished reviewing the following examples, be sure to read Chapter 6, “Using Reports to Query and Update Data Sources,” which provides more information about creating custom reports and server-side scripts that enable users to dynamically update attribute data sources. This technique is illustrated in the Municipal Application, which begins on page 140.

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 Autodesk MapGuide 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.

To create a custom redline application

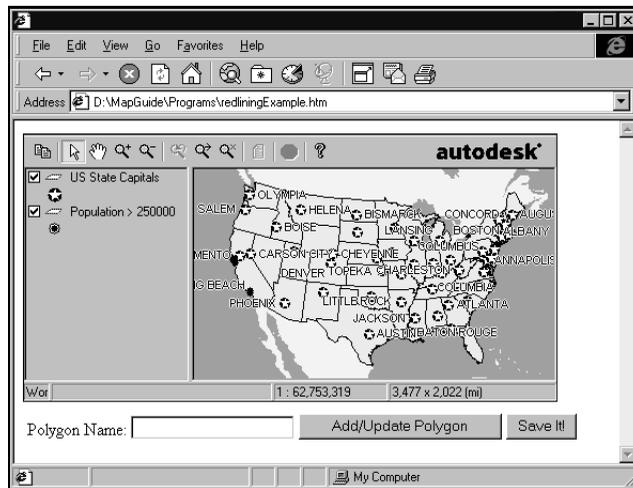
- 1 Use the `MGMap.createLayer` method to create the redline layer, or, if the layer already exists, you can access it with the `MGMap.getMapLayer` method.
- 2 Use the `MGMapLayer.createMapObject` method to add an empty redline object to the layer.
- 3 Use the `MGMapObject` add methods to add one or 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 `MGMap.saveMWF` method to save the map file to an MWF on the user's machine or a network server. `MGMap.saveMWF` is not available for Autodesk MapGuide Viewer, Java Edition.

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 "SDF Component Toolkit Applications" on page 171.

Redlining Example Code

The following example shows one way to write a simple redlining application. It 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"
OnClick="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 = getMap();

    // 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 `onDigitizedPolygon` 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:

onDigitizedPolygon 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 layer = map.getMapLayer("My Redline Layer");
    if (layer == null)
        layer = 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)
```

onDigitizedPolygon Function (continued)

```
var obj = layer.getMapObject(formText);
if (obj == null)
    var obj = layer.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 MGMap.saveMWF method, and saves the map to
the specified path:
function save_it()
{
var fName = "c:\\My Documents\\my_map.mwf";
var password = prompt("Please enter a password.", "");

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

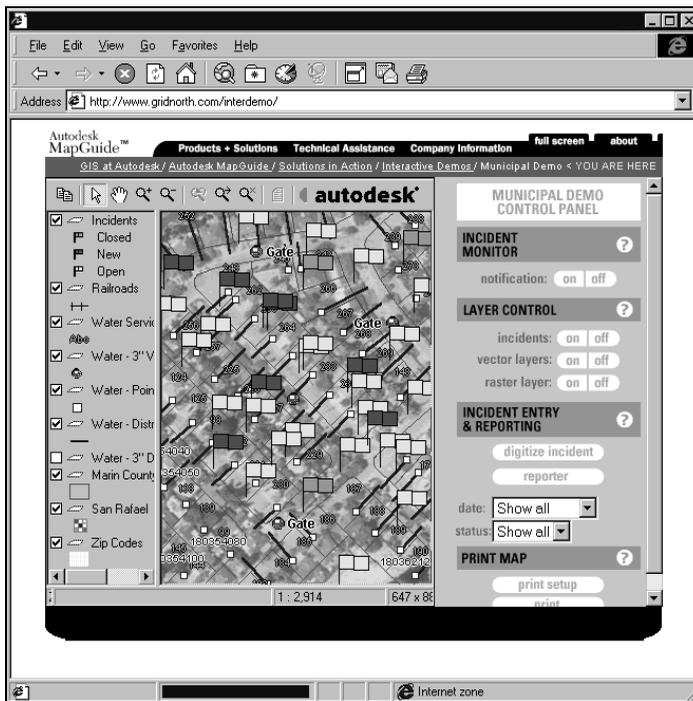
The example above is a very simple application designed to illustrate redlining. However, your application can have more features, such as allowing users to add other primitives besides polygons. Also, you can exert more control over how the primitives appear on screen or to query the state of existing 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`, and so on). Also look up the `MGMapdigitize` 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 Municipal Application demonstrates how you can 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.

At the left of the window, the standard legend allows you to turn layers on and off and select them. At the right of the window the application includes Incident Monitor, where users are notified if there are problems with the water distribution system, and Layer Control, which allows you to turn off the incident layer, turn off all vector layers at the same time, and turn off the raster layer. These controls are useful for finding information quickly. Lastly, under Incident Entry and Reporting, the digitize incident and reporter buttons allow the user to add an incident to the map and generate reports about selected features.



Municipal application

Municipal Application Example 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 Example Code

```
<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 map;

// 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: getMap()
// Description: Get appropriate map object for IE or Netscape
// Arguments:   none
// Returns:    map object
// ===== //
function getMap()
{
    // Get appropriate MGMap object; type depends on browserId value
    if (browserId == 1)
        map = top.main.document.embeds[0]; // Netscape map object
    else if (browserId == 2)
        map = top.main.document.map;      // IE map object
    else
        map = null;                       // none if other browser
    return map;                           //return map object
}
```

Municipal Application Example Code (continued)

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

// ===== //
// 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; 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 Example Code (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)
{
    map = getMap(); // Get instance of MGMap
    if (map.isBusy() == false) // If Autodesk MapGuide Viewer is not busy...
    {
        // ...get "Incidents" layer
        var mapLayer1 = map.getMapLayer("Incidents");

        // If function was called with 'Off', make the
        // layer invisible; otherwise make the layer visible
        if (type == 'Off')
            mapLayer1.setVisibility(false);
        else
            mapLayer1.setVisibility(true);

        // Tell the Autodesk MapGuide Viewer to rebuild layer
        // when map is refreshed; then refresh map
        mapLayer1.setRebuild(true);
        map.refresh();
    }
    // If Autodesk MapGuide Viewer is busy, don't do the stuff above;
    // instead, display alert
    else
        alert("The Autodesk MapGuide 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)
{
```

Municipal Application Example Code (continued)

```
map = getMap(); // Get instance of MGMap
if (map.isBusy() == false) // If Autodesk MapGuide Viewer is not busy...
{
    // ...assign a bunch of layers to a bunch of variables
    var mapLayer2 = map.getMapLayer("Population > 100000");
    var mapLayer3 = map.getMapLayer("Population > 50000");
    var mapLayer4 = map.getMapLayer("Population > 500");
    var mapLayer5 = map.getMapLayer("Population > 0");
    var mapLayer6 = map.getMapLayer("Railroads");
    var mapLayer7 = map.getMapLayer("Interstates");
    var mapLayer8 = map.getMapLayer("Highways");
    var mapLayer9 = map.getMapLayer("Major Roads");
    var mapLayer10 = map.getMapLayer("Minor Roads");
    var mapLayer11 = map.getMapLayer("Water Service Acct.");
    var mapLayer12 = map.getMapLayer("Water - 3 inch Valves");
    var mapLayer13 = map.getMapLayer("Water - Point of Service");
    var mapLayer14 = map.getMapLayer("Water - Distribution");
    var mapLayer15 =
        map.getMapLayer("Water - 3 inch Distribution");

    var mapLayer16 =
        map.getMapLayer("Marin County Land Parcels");
    var mapLayer17 = map.getMapLayer("Population Density");
    var mapLayer18 = map.getMapLayer("ZIP Codes");
    var mapLayer19 = map.getMapLayer("Counties");

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

Municipal Application Example Code (continued)

```
// Refresh the map
map.refresh();

// End the if statement that verified not busy
}
// If Autodesk MapGuide Viewer is busy, don't do the stuff above;
// instead, display alert
else
    alert("The Autodesk MapGuide 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)
{
    map = getMap();           // Get instance of MGMap

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

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

Municipal Application Example Code (continued)

```
// instead, display alert
else
    alert("The Autodesk MapGuide 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.
//
// 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 getMap(), because entire function
    // varies by browser

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

        // If Autodesk MapGuide 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 (map.isBusy() == false)
            map.digitizePoint(parent.rightempty.document.obs);
        else
            alert("The Autodesk MapGuide Viewer is busy. Please try again
                in a few seconds.");
    }
    // If Internet Explorer...
    if (browserId == 2)
```

Municipal Application Example Code (continued)

```
{

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

    // If Autodesk MapGuide Viewer is not busy, call
    // digitizePoint() API method;
    // otherwise display alert
    if (map.isBusy() == false)
        map.digitizePoint();
    else
        alert("The Autodesk MapGuide Viewer is busy. Please try again
            in a few seconds.");
}

}

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

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

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

Municipal Application Example Code (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()
{
    map = getMap();           // Get instance of MGMap

    // If Autodesk MapGuide Viewer is not busy...
    if (map.isBusy() == false)
    {
        // If user has old copy of Autodesk MapGuide Viewer
        // display alert only
        var ApiVersion = map.getApiVersion();
        if (ApiVersion < 6.0)
        {
            alert("This control uses the latest technology in the\n
                Autodesk MapGuide Viewer API. Please \n
                download the latest Autodesk MapGuide Viewer from\n
                the Autodesk MapGuide\n
                Web site (www.autodesk.com/mapguideviewerdownload).");
        }
        // If user has recent copy of Autodesk MapGuide Viewer (API version 6.0
        // 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 variable

```

Municipal Application Example Code (continued)

```
if (temp == 'showAll')
{
    var whereClause1 = "Status is not null";
}
// 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')
```

Municipal Application Example Code (continued)

```
        {
            whereClause2 =
                "ReportDate <= #04/01/99# AND ReportDate >= #03/02/99#"
        }
        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 'mapDatabaseSetup'
        var mapLayer = map.getMapLayer("Incidents");
        var mapLayerSetup = mapLayer.getLayerSetup();
        var mapDatabaseSetup = mapLayerSetup.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.
        mapDatabaseSetup.setWhereClause(whereClause);
        map.refresh();
    }
}

// ===== //
// Function: pageSetup()
// Description: Displays the page setup dialog (called by
// the 'Print Setup' button, under 'Print Map')
// Arguments: none
// Returns: nothing
// ===== //
function pageSetup()
{
    map = getMap();           // Get instance of MGMap

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

// ===== //
// Function printMap()
```

Municipal Application Example Code (continued)

```
// Description: Displays Print dialog (called by the
// 'Print' button, under 'Print Map')
// Arguments: none
// Returns: nothing
// ===== //
function printMap()
{
    map = getMap();           // Get instance of MGMap

    // If Autodesk MapGuide Viewer is not busy, display Print dialog;
    // otherwise display alert
    if (map.isBusy() == false)
        map.printDlg();
    else
        alert("The Autodesk MapGuide 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 CELLSPACING=0 CELLPADDING=0 BORDER=0 WIDTH=100%>
<TR>
<TD VALIGN=top width=100%>

    <!-- Begin table 2 -->
    <TABLE CELLSPACING=2 CELLPADDING=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 CELLSPACING=2 CELLPADDING=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>

```

Municipal Application Example Code (continued)

```
<IMG SRC="MENU/incmon.gif" WIDTH=56 HEIGHT=29 BORDER=0><BR>
</TD>
</TR>
<TR>
<TD VALIGN=MIDDLE ALIGN=CENTER>

    <-! 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"
```

Municipal Application Example Code (continued)

```
        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>
</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>
```

Municipal Application Example Code (continued)

```
<!-- End table 5 -->

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

Municipal Application Example Code (continued)

```
</TR>
</TABLE>
<--! End table 10 -->

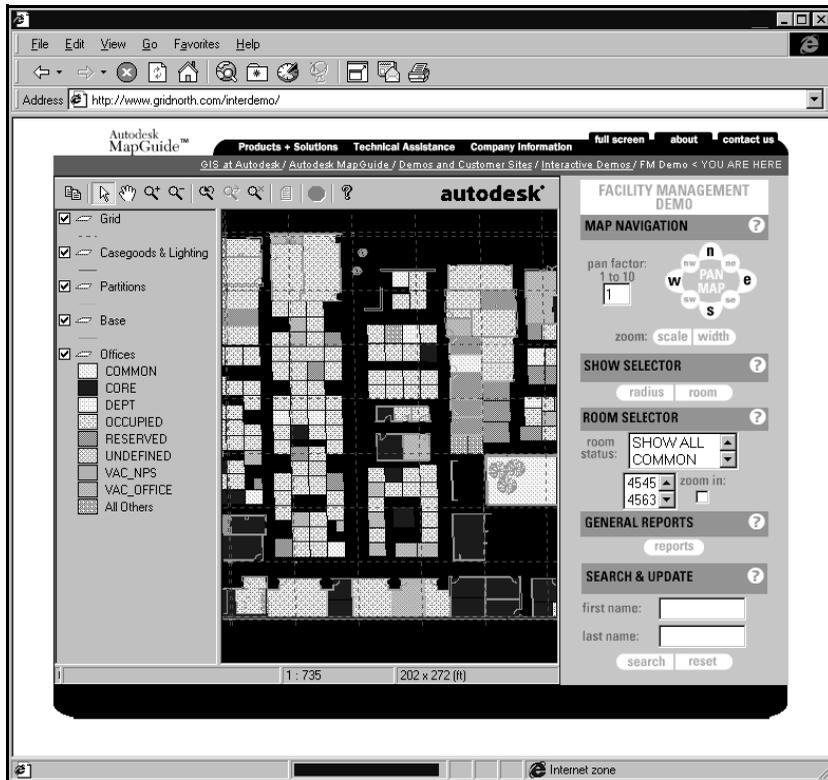
<--! Begin table 11 -->
<TABLE CELLPADDING=2 CELLSPACING=3 BORDER=0 WIDTH=100%>
<TR>
<TD BGCOLOR="#9c9c63">
<A HREF="app98help.cfm/#mapprn"><IMG SRC="MENU/qs.gif"
  WIDTH=21 HEIGHT=18 BORDER=0 align=right></A>
<IMG SRC="menu/prnmap.gif" WIDTH=64 HEIGHT=18 BORDER=0><BR>
</TD>
</TR>
<TR>
<TD VALIGN=MIDDLE 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 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 at the right of the window. 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

Facilities Management Application Example 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 at 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.

Facilities Management Application Example Code

```
<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 map;

// 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: getMap()
// Description: Get appropriate map object for IE or Netscape
// **Same concept as getMap() **
// Arguments:    none
// Returns:      map object
// ===== //
function getMap()
{
    // Get appropriate MGMap object; type depends on browserId value
    if (browserId == 1)
        map = top.main.document.embeds[0]; // Netscape map object
    else if (browserId == 2)
        map = top.main.document.map;      // IE map object
```

Facilities Management Application Example Code (continued)

```
    else
        map = null; // none if other browser
    return map; //return map object
}

// ===== //
// Function: Pan(direction)
// Description: pans in the specified direction
// Arguments: direction
// Return: nothing
// ===== //
function Pan(direction)
{
    // Get MGMap object
    var map = getMap();
    map.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 = map.getWidth("M");
            delta = (scrollfactor/10) * delta;
        // Compute center point of map in Mapping Coordinate System (MCS)
        var xyPt = map.lonLatToMcs(map.getLon(), map.getLat());
        // Convert delta from Meters to MCS units.
        var MCStoMeters = map.getMCSSToMeters();
        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)
        }
    }
}
```

Facilities Management Application Example Code (continued)

```
        xyPt.setY(xyPt.getY() + delta)
    }
    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 = map.getScale();
    map.zoomScale(xyPt.getY(), xyPt.getX(), myScale);
    map.setAutoRefresh(true);
    map.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 map = getMap();
    map.zoomWidth(lat, lon, 400, "FT");
}

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

Facilities Management Application Example Code (continued)

```
        map.viewReportsDlg();
    }
}

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

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

/*+++++
+ Function:      selRadiusMode()                            +
+ Description:   changes the selection mode to select by radius +
+ Arguments:    none                                       +
+ Return:       none                                       +
+++++*/
function selRadiusMode(){
    var map = getMap();
    map.selectRadiusMode();
}

/*+++++
+++++*/
```

Facilities Management Application Example Code (continued)

```
+ Function:      ObjSelChanged()                               +
+ Description:   on change of selection, highlight room selected +
+ Arguments:    none                                         +
+ Return:       none                                         +
+******/
function ObjSelChanged()
{
    // Get MGMap object
    var map = getMap();
    if (map.isBusy() == false){
        var selOptions = document.Selection.roomnum.options;
        var collection = map.createObject("MGCollection");
        var sel = map.getSelection();
        var mapLayer = map.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 obj = mapLayer.getMapObject(selOptions[i].value);
                if (obj != null) {
                    collection.add(obj);
                }
            }
        }
        sel.clear();
        if (collection.size() > 0){
            sel.addObjectsEx(collection, false);
        }
        var zoomCheck = document.Selection.ZoomOption.checked;
        if (zoomCheck == true){
            map.zoomSelected();
        }
    }
}

/******
+ Function:      showOccupancy()                               +
+ Description:   on change of selection, show occupancy type   +
+ Arguments:    none                                         +
+ Return:       none                                         +
+******/
function showOccupancy()
{
    // Get MGMap object
    var map = getMap();
    if (map.isBusy() == false){
        var selValue = document.Selection.Status.options;
        for (var i=0; i < selValue.length; i++) {
```

Facilities Management Application Example Code (continued)

```
        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 mapLayer = map.getMapLayer("Offices");
    mapLayer.setSQLWhere(whereClause);
    map.refresh();
}

}

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

/*+++++
+ Function      openSearchWind()                            +
+ Description:  used to launch the search popup window          +
+ Arguments:   none                                           +
+ Return:      none                                           +
+++++*/
function openSearchWind(){
    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. \n
            Or, select a room before continuing.");
    }
}
```

Facilities Management Application Example Code (continued)

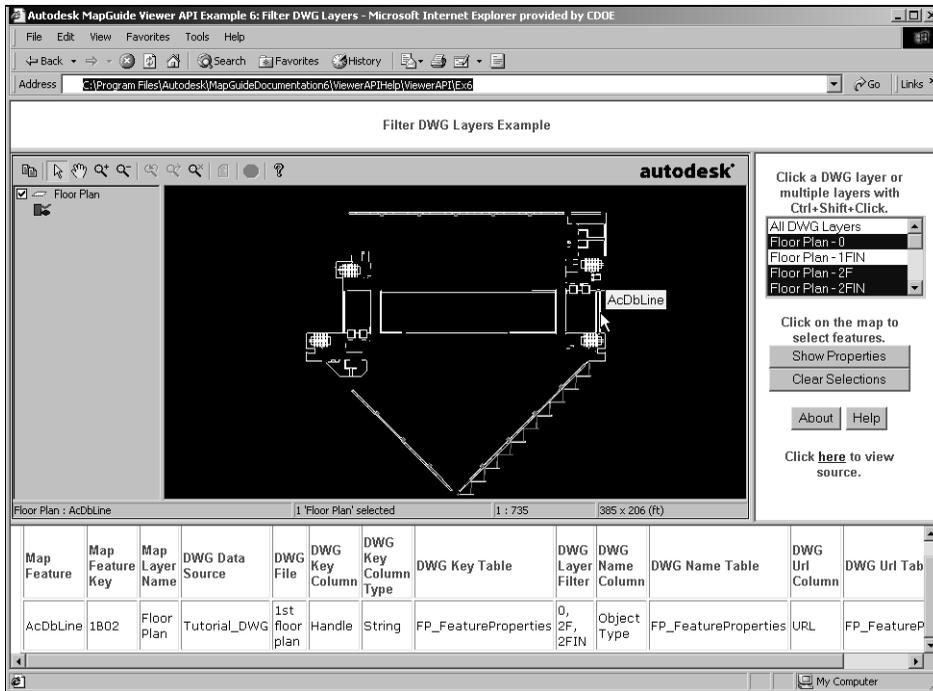
```
    else{
        SearchWindow = window.open("Search.cfm?FirstName="
            +FirstName+"&LastName="+LastName+ "&Room="+RoomSelected,
            "SearchWindow", "toolbar=no,width=350,height=205,directories=no,
            status=no,scrollbars=no,resize=yes,menubar=no")
    }
}

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

DWG Filtering Application

This example demonstrates filtering Autodesk DWG layers and selecting map features. Autodesk DWG is a worldwide standard across vertical industries, such as architectural design, and facilities planning and maintenance. For a working version of this application, choose Help > Contents > Examples Advanced > DWG Filtering Example in the *Autodesk MapGuide Viewer API Help*.

As shown in the following illustration, the list box to the right of the map shows the DWG filters that can be applied to an architectural map of a floor plan. When a user selects one or more filters from the list box, the corresponding Autodesk DWG layer(s) is displayed on the map. The user can also select one or more features on the map and display properties, including names and keys, and DWG layer properties for the current map layer.



DWG Filtering application

Understanding Layers in Autodesk MapGuide

When working with DWG data especially, it is important to understand the difference between Autodesk DWG layers, Autodesk MapGuide map layers, and Autodesk DWG map layers. These are not the same and are defined as follows:

Layer Type	Definition
Autodesk DWG layer	A single layer within an Autodesk drawing file (Autodesk DWG).
Autodesk MapGuide map layer	A single map layer defined within an Autodesk MapGuide map window file (MWF). There can be several types of Autodesk MapGuide map layers, including ones sourced from SDF, SHP, DWG, and so on.
Autodesk DWG map layer	A single Autodesk MapGuide map layer that is sourced from an Autodesk DWG or Autodesk Map project drawing.

Changing Map Layer Data Source Properties

When using Autodesk MapGuide Viewer API methods that set Autodesk MapGuide map layer data source properties, including those for Autodesk DWG data, keep in mind that the changes you make can affect the validity of other data source properties for that layer. For example, if you changed the DWG data source using the `MGDwgDataSources.setDataSource` method (or the `MGDwgDataSources.DataSource` property), this change could make existing values for other properties (such as the Autodesk DWG file) invalid since they might not exist in the new data source.

DWG Filtering Application Example Code

This example illustrates some basic tasks an application can perform when working with DWG data. It primarily focuses on setting DWG filters and getting DWG data, but it can easily be modified to change other Autodesk MapGuide map layer properties, such as the DWG file, the DWG data source, and so on.

In this example, DWG processing is performed on a map with a single Autodesk MapGuide map layer called `Floor Plan`. Processing is performed by two functions: `selChanged()` and `showProperties()`. The operational flow of these functions is described in the following sections. To see the full source code for this example, choose `Help > Contents > Examples Advanced > Filter DWG Layers` in the *Autodesk MapGuide Viewer API Help*. When the application loads, click the view source link in the right frame.

Selection Changed Function (`selChanged`)

The following are excerpts from the `selChanged()` function. This function is called when a user selects or deselects one or more DWG filters from a list box displayed on the map page. This function sets the layer filter of the DWG map layer to layers selected by the user. The basic operational flow of this function is as follows:

- 1 Use the `getMap` function, the current map is assigned to the variable `map`.
- 2 Use the `map.getMapLayer` method to get the map's single map layer named `Floor Plan`. Assign the map layer to a variable called `mapLayer`.
- 3 Get the DWG data source object for the layer by first getting the layer setup using `mapLayer.getLayerSetup`, and then using the layer setup to get the DWG data sources object of the layer using `mapLayerSetup.getDwgDataSources`.
- 4 Use `map.createObject("MGCollection")` to create a collection object called `mapDwgLayerFilters`. This object will hold filters selected by the user from the filters list box.
- 5 Iterate through list box selections and add each to the collection object `mapDwgLayerFilters`.
- 6 Create a comma-delimited string named `filterList` containing selected filters. Note that the `setLayerFilter` method used next expects a comma-delimited string of filters.

- 7 Set the layer filter from the selected filter list using the `dwgMapLayerDataSource.setLayerFilter(filterList)` method.
- 8 Refresh the map using the `map.refresh` method.

DWG Filtering Application `selChanged` Example Code

```
// Selection Changed Function
// Called whenever items are selected or deselected in the list box
function selChanged()
{
    var map = getMap();
    // Use the single map layer Floor Plan
    var mapLayer = map.getMapLayer("Floor Plan");
    // Get DWG data source object for the layer
    var mapLayerSetup = mapLayer.getLayerSetup();
    var dwgMapLayerDataSource = mapLayerSetup.getDwgDataSources();
    // Create a collection object to hold filters selected from list box
    var mapDwgLayerFilters = map.createObject("MGCollection");
    // Iterate through list box selections and add each to the collection
    var selOptions = document.forms[0].filters.options;
    for (var i = 0; i < selOptions.length; i++)
    {
        if (selOptions[i].selected)
        {
            var selectedFilter = selOptions[i].value;
            if (selectedFilter != null)
                mapDwgLayerFilters.add(selectedFilter);
        }
    }
    // Create comma-delimited string containing selected filters.
    var filterList = "";
    for (var i = 0; i < mapDwgLayerFilters.size(); i++)
    {
        var filter = mapDwgLayerFilters.item(i);
        if (i == mapDwgLayerFilters.size() - 1)
            filterList = filterList + filter;
        else
            filterList = filterList + filter + ", ";
    }
    // Set the layer filter from the selected filter list
    dwgMapLayerDataSource.setLayerFilter(filterList);
    map.refresh();
}
```

Show Properties Function (showProperties)

The following are excerpts from the `showProperties()` function. This function gets and displays properties of selected map features in a table in the output frame of the Web page, including DWG properties associated with the DWG map layer. The basic operational flow of this function is as follows:

- 1 Write JavaScript and style sheet selection code to output frame.
- 2 Write output table headings.
- 3 Get the map and selected features using the `getMap` function and the `map.getSelection`, and `sel.getMapObjectsEx(null)` methods.
- 4 Iterate through the selected map features (map objects) and write the feature(s) properties to the output table. In this example, we get the feature name and key using the `MGMapObject.getName` and `getKey` methods.
- 5 Get the layer name for current map feature. To do this, we first get the map layer of the selected feature using the `MGMapObject.getMapLayer` method. We then use the `MGMapLayer.getName` method.
- 6 Get the map layer and its DWG data source object. To do this we first get the map layer using `map.getMapLayer` with the layer name, then get the layer setup using `mapLayer.getLayerSetup` method. Then, we get the DWG data sources object of the layer using `mapLayerSetup.getDwgDataSources`.
- 7 Using `MGDwgDataSources` methods, we get the DWG properties of the map layer, including `getDataSource`, `getDwg`, `getKeyColumn`, `getKeyColumnType`, and `getKeyTable`.
- 8 Next, using the `getLayerFilter` method, we get the DWG layer filters used to display the current map view, and indicate that if no filter was applied if `layerFilter` is null.
- 9 Finally, we complete the table by listing the remaining DWG properties using the `MGDwgDataSources` methods `getNameColumn`, `getNameTable`, `getUrlColumn`, and `getUrlTable`.

DWG Filtering Application showProperties Example Code

```
// Show Properties Function
// Shows properties of the selected map feature(s), including DWG properties of
the layer
function showProperties()
{
    // Write javascript and style sheet selection code to output frame
    ...
// Write table headings
parent.outputframe.document.write("<TABLE BORDER=1 WIDTH='100%'>");
parent.outputframe.document.write("<TR>");
```

DWG Filtering Application showProperties Example Code (continued)

```
parent.outputframe.document.write("<TD>");
parent.outputframe.document.write("<h3>Map Feature</h3>");
parent.outputframe.document.write("</TD>");
...
parent.outputframe.document.write("</TR>");

// Get the map and selected features
var map = getMap();
var sel = map.getSelection();
var mapObjects = sel.getMapObjectsEx(null);
// Write the feature(s) properties to the table
for (var i = 0; i < mapObjects.size(); i++)
{
    parent.outputframe.document.write("<TR>");
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(mapObjects.item(i).getName());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(mapObjects.item(i).getKey());
    parent.outputframe.document.write("</TD>");
    // Get the layer name for current map feature
    var layerName = mapObjects.item(i).getMapLayer().getName();
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(layerName);
    parent.outputframe.document.write("</TD>");
    // Get the map layer and its DWG data source object
    var mapLayer = map.getMapLayer(layerName);
    var mapLayerSetup = mapLayer.getLayerSetup();
    var dwgMapLayerDataSource = mapLayerSetup.getDwgDataSources();
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write
        (dwgMapLayerDataSource.getDataSource());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(dwgMapLayerDataSource.getDwg());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(dwgMapLayerDataSource.getKeyColumn());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>>");
    parent.outputframe.document.write
        (dwgMapLayerDataSource.getKeyColumnType());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>");
    parent.outputframe.document.write(dwgMapLayerDataSource.getKeyTable());
    parent.outputframe.document.write("</TD>");
    parent.outputframe.document.write("<TD>");
    // Get the DWG layer filters used to display the current map view.
    // Indicate if no filter was applied.
```

DWG Filtering Application showProperties Example Code (continued)

```
var layerFilter = dwgMapLayerDataSource.getLayerFilter();
if (layerFilter == "")
    parent.outputframe.document.write ("No filter applied");
else
    parent.outputframe.document.write (layerFilter);
parent.outputframe.document.write ("</TD>");
parent.outputframe.document.write ("<TD>");
parent.outputframe.document.write
(dwgMapLayerDataSource.getNameColumn());
parent.outputframe.document.write ("</TD>");
parent.outputframe.document.write ("<TD>");
parent.outputframe.document.write (dwgMapLayerDataSource.getNameTable());
parent.outputframe.document.write ("</TD>");
parent.outputframe.document.write ("<TD>");
parent.outputframe.document.write (dwgMapLayerDataSource.getUrlColumn());
parent.outputframe.document.write ("</TD>");
parent.outputframe.document.write ("<TD>");
parent.outputframe.document.write (dwgMapLayerDataSource.getUrlTable());
parent.outputframe.document.write ("</TD>");
parent.outputframe.document.write ("</TR>");
}
parent.outputframe.document.write ("</TABLE>");
parent.outputframe.document.write ("</CENTER>");
parent.outputframe.document.write ("</BODY>");
parent.outputframe.document.close();
}
```

SDF Component Toolkit Applications

This section presents example applications using the SDF Component Toolkit: one Active Server Page (ASP) example and three Visual Basic (VB) examples.

The SDF Component Toolkit is a set of COM objects for reading and writing Spatial Data Files (SDFs), as well as their supporting Spatial Index Files (SIFs) and Key Index Files (KIFs). The spatial data displayed in Autodesk MapGuide maps (map features such as roads, countries, buildings, and so on) is contained in SDFs. SDF data is indexed in SIFs and KIFs. You can access SDF Component Toolkit objects in development environments such as C++, Visual Basic, VBA, VBScript, Java, JScript, ASP, CGI, and ColdFusion.

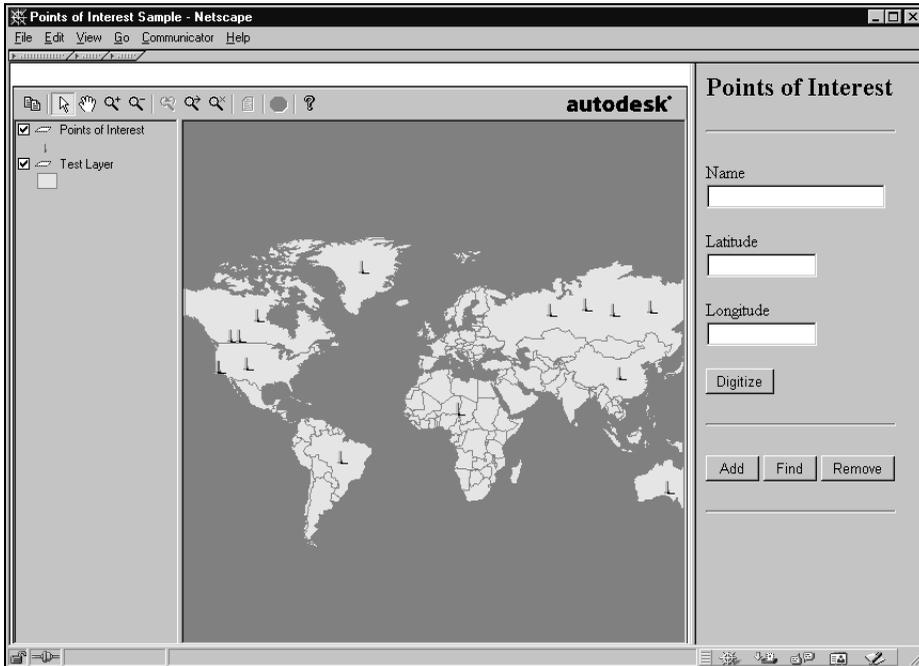
For more information about the SDF Component Toolkit, refer to the SDF Component Toolkit Help, a help file (*SDFCOMTK.HLP*) located in the *\Help* directory in your *SDF Component Toolkit* directory.

Updating SDF Files—an ASP Example

You can use the 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.

You can use the SDF Component Toolkit to add points, polylines, or polygons to an SDF file when the user clicks the map. Note that Autodesk MapGuide contains a similar functionality called *redlining*. This means that users can add features to the map without using the SDF Component Toolkit (see “Custom Redlining Application” on page 136). 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 file as its data source will display the new points.

This example 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, which will also be added to the corresponding SDF data source.



SDF Component Toolkit Points of Interest application

Notice that the map is at the left and the controls are at 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, type the name of the point and click the Find button. The map zooms to the point. To select the point, either click it or 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 until the point appears. To remove the point from the map, type the name of the 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:

***maps_poi.htm* Example Code**

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

- 1 Embed the map.
- 2 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 *MapGuideObserver6.class* file for use by Netscape.
- 3 Define a function that takes the point from the observers and updates the text boxes with the point's coordinates.

***map.htm* Example Code**

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

<SCRIPT LANGUAGE="VBScript">
// Send onDigitizedPoint events from Autodesk MapGuide Viewer 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;
```

map.htm Example Code (continued)

```
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")
{
document.write("<APPLET CODE=\"MapGuideObserver6.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 MapGuideObserver6 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="">
```

map.htm Example Code (continued)

```
<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=
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>
```

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. The ASP code in `poi.asp`, which drives the controls in the right-hand frame, 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 110.

poi.asp Example Code

```
<%@ 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.
```

poi.asp Example Code (continued)

```
// 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");

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.MgSdfObjectGeometry.1");
var sdfSegment =
Server.CreateObject("Autodesk.MgSdfObjectGeometrySegment.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);
```

poi.asp Example Code (continued)

```
sdfObject.SetGeometry(0, sdfGeometry);
sdfObject.Name = pointName;
sdfObject.Key = pointName;
sdfObject.Url = "";

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

poi.asp Example Code (continued)

```
zoomToLon = sdfPoint.X;
zoomToLat = sdfPoint.Y;
clearVars();
actionOp = "ZoomToPoint"; // Zooms to the point on the map
msgText = "Zooming to point.";
}
else
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%>"
```

poi.asp Example Code (continued)

```
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%>"
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";
```

poi.asp Example Code (continued)

```
function getMap()
{
if (ver == "n4")
return parent.mapFrame.document.map;
else
return parent.mapFrame.map;
}
// Following is the function called by the Digitize button. If the
// browser is Navigator, it will send the onDigitizedPoint event to
// the MapGuideObserver6.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")
getMap().digitizePoint(parent.mapFrame.document.obs);
else
getMap().digitizePoint();
}

function updateMap()           // Updates the map after an Add or Remove
{
getMap().getMapLayer("POI").setRebuild(true);
getMap().refresh();
}

function zoomToPoint()        // Zooms to the point after a Find
{
getMap().zoomWidth(<%=zoomToLat%>, <%=zoomToLon%>, 1000, "Mi");
getMap().refresh();
}
</SCRIPT>

<%
//When the resulting HTML 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();");
}
```

poi.asp Example Code (continued)

```
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 page at www.autodesk.com/mapguidedemo to see more applications and add your own applications to share with others.

Converting to an SDF File—a Visual Basic Example

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. This example is written in Visual Basic.

ConvertSDF Example Code

```
VERSION 6.00  
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  
    End  
    Begin VB.CommandButton btnConvert
```

ConvertSDF Example Code (continued)

```
    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"
    Top          = 480
    Width        = 4212
```

ConvertSDF Example Code (continued)

```
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
    cdOpen.ShowSave
    txtSdfName.Text = cdOpen.FileName
End Sub
```

ConvertSDF Example Code (continued)

```
ErrorHandler:
    'Cancel was selected
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub

Private Sub btnBrowseTxt_Click()
On Error GoTo ErrorHandler

    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

ErrorHandler:
    '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 ErrorHandler

    'Open the input text file
    Open txtTxtName.Text For Input As #1
    'Open the sdf file in read-only mode
    oTlkt.Open txtSdfName.Text, sdfOpenUpdate Or sdfCreateAlways, True
    'Indicate update process
```

ConvertSDF Example Code (continued)

```
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
'Wind up
oTlkt.EndUpdate
oTlkt.Close
```

ConvertSDF Example Code (continued)

```
Close #1

ShowMessage txtTxtName.Text & " converted to " & txtSdfName.Text
ShowMessage "Total features converted: " & j
Exit Sub

ErrorHandler:
'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 ErrorHandler

'Open the sdf file in read-only mode
oTlkt.Open txtSdfName.Text, sdfOpenUpdate Or sdfOpenExisting, True
'Begin spatial search for polylines
oTlkt.BeginKeyIndexSearch (Key)
'Get first feature having key
Set oObj = oTlkt.SearchToNextObject()
'End search
oTlkt.EndSearch
```

ConvertSDF Example Code (continued)

```
objFound = Not (oObj Is Nothing)
If Not (oObj Is Nothing) Then
    ShowMessage "Following feature is deleted"
    ShowMessage "Feature: " & i & " " & GetObjectTypeString(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

ErrorHandler:
'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

Function GetObjectTypeString(ObjType As SdfObjectType) As String
    Select Case ObjType
        Case sdfPointObject:
            GetObjectTypeString = "POINT"
        Case sdfPolygonObject:
```

ConvertSDF Example Code (continued)

```
        GetObjectTypeString = "POLYGON"
    Case sdfPolylineObject:
        GetObjectTypeString = "POLYLINE"
    Case sdfPolyPolylineObject:
        GetObjectTypeString = "POLYPOLYLINE"
    Case sdfPolyPolygonObject:
        GetObjectTypeString = "POLYPOLYGON"
End Select

End Function
```

Getting SDF File Information—a Visual Basic Example

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. This example is written in Visual Basic.

SDFInfo Example Code

```
VERSION 6.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0"; "RICTX32.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
    Begin VB.CommandButton btnSrchKey
        Caption       = "Key Search"
        Height        = 288
        Left          = 4440
    End
End
```

SDFInfo Example Code (continued)

```
    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
    Width         = 4212
End
Begin VB.CommandButton btnBrowse
    Caption       = "Browse"
```

SDFInfo Example Code (continued)

```
    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
    'Just exit after resetting errhandler
    On Error GoTo 0
End Sub
```

SDFInfo Example Code (continued)

```
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 & _
        "Max LAT: " & oTlkt.BoundingBox.maxY & vbCrLf & _
        "Max LON: " & oTlkt.BoundingBox.maxX

    ShowMessage "SDF Extents: " & vbCrLf & strMsg
End Sub
```

SDFInfo Example Code (continued)

```
'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(sdfPointClass)
ShowMessage "Contains Polylines: " &
oTlkt.ContainsObjectClass(sdfPolylineClass)
ShowMessage "Contains Polygons: " &
oTlkt.ContainsObjectClass(sdfPolygonClass)

'Close the toolkit
oTlkt.Close

Exit Sub

ErrorHandler:
'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
Dim strMsg As String
Dim i As Long
Dim objFound As Boolean
```

SDFInfo Example Code (continued)

```
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 & " " & GetObjectTypeString(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

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
```

SDFInfo Example Code (continued)

```
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 & " " & GetObjectTypeString(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

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
```

SDFInfo Example Code (continued)

```
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 & " " & GetObjectTypeString(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

Sub ShowMessage(Msg As String)
```

SDFInfo Example Code (continued)

```
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—a Visual Basic Example

The CopySDF example shows how to open an existing SDF file and write its features to a new SDF file. It is written in Visual Basic.

CopySDF Example Code

```
VERSION 6.00
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0"; "RICTX32.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 btnCopy
    Caption           = "Copy"
    Height            = 288
    Left              = 4440
    TabIndex          = 6
    Top               = 960
    Width             = 1212
```

CopySDF Example Code (continued)

```
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
    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
```

CopySDF Example Code (continued)

```
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
  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
```

CopySDF Example Code (continued)

```
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
    '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)
```

CopySDF Example Code (continued)

```
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

ErrorHandler:
'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
```

CopySDF Example Code (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
```


Index

A

- accessing maps
 - from ActiveX Control 32
 - from Java Edition 35
 - with HTML 35
 - with Java 38
 - with JavaScript/JScript 35
 - from Plug-In 32
 - with applet for Plug-In 40
- accessing secure data 43
- Active Server Pages (ASP)
 - about 109
 - and Autodesk MapGuide LiteView 26
 - creating reports with 109
 - defined 88
 - displaying data 115
 - listing database contents with 109
 - listing file contents 111
 - querying data 14, 115
 - SDF Component Toolkit example 171
 - updating data 14
- ActiveX Control 11
 - detecting with CODEBASE 27
 - displaying a map in 20
 - embedding a map 22
 - installing on client machines 27
 - linking to a map 20
- adding
 - custom page elements 68
 - map layers 53
- applet
 - accessing JavaScript 42
 - for Plug-In 40
 - for Plug-In event handler 83
 - peer for Java Edition 38
 - wrapper for Java Edition 24, 38
 - See also* Java, Java Edition
- application examples
 - DWG filtering 164
 - facilities management 156
 - municipal 140
 - redlining 136
 - SDF Component Toolkit 171-72, 181, 188, 196
- applications 11, 136
 - advanced 15
 - as Java wrapper for Java Edition 38
 - creating 15, 17
 - debugging 48
 - simple 15
 - stand-alone 19
- argument type checking 48
- ASP. *See* Active Server Pages
- attribute data
 - displaying for selected features (reports) 85, 86
 - listing with ColdFusion 89, 92
 - updating dynamically 86, 102, 125
- Autodesk DWG. *See* DWG 8
- Autodesk MapGuide Author
 - creating popup menus with 108, 132
 - creating reports with 92, 103, 116, 126
- Autodesk MapGuide LiteView 26
- Autodesk MapGuide Server security 43
- Autodesk MapGuide Viewer ActiveX Control. *See* ActiveX Control
- Autodesk MapGuide Viewer Plug-In. *See* Plug-In
- Autodesk MapGuide Viewer, Java Edition. *See* Java Edition
- Autodesk MapGuide Viewer. *See* Viewer
- Autodesk MapGuide Web site 13
- autoRefresh 46

B

- bitmap support 8
- browsers supported 9, 18
- busy state
 - about 43
 - detecting change in 46
 - handling 43

C

- CFM files 89, 92, 102
- checking argument types 48

- ColdFusion 10
 - about 89
 - and Autodesk MapGuide LiteView 26
 - creating reports with 85, 86, 89, 142, 146, 149
 - defined 88
 - listing database contents with 89, 92, 102
 - querying data 14, 85
 - template files 89, 92, 102
 - updating data 14, 85
- coordinates of map features 58
- counting
 - map features 62
 - map layers 50
- creating Viewer API applications 15
- custom page elements 68

D

- data
 - attribute, listing with ColdFusion 89, 92
 - attribute, updating dynamically 86, 102, 125
 - querying 14
 - reports 85, 86
 - security 43
 - updating 14
 - updating via map 102
 - updating via the map 86, 125
- databases, listing contents
 - with Active Server Pages (ASP) 109
 - with ColdFusion 89, 92, 102
- debugging 48
 - enabling/disabling errors 48
 - handling errors 47
- detecting
 - busy state 46
 - map busy state change 46
 - map refresh 46
- digitizing
 - circles 61
 - points 140
- displaying maps 14
 - ActiveX Control 20
 - area of 21
 - for Plug-In 20
 - in frame 21
- downloading required software 25, 34, 39
- drawing file. *See* DWG
- DSNs
 - working with 90, 102, 112, 116, 125
 - See also* OLE DB
- DWG 8, 164
 - application examples 164
 - DWG layer, defined 165
 - DWG map layer, defined 165

- DWG (*continued*)
 - filtering layers 164
 - MapGuide map layer, defined 165
- Dynamic Authoring Toolkit 10

E

- embedding
 - Java Edition in an HTML page 35
 - map 18
 - for ActiveX Control 22
 - for Java Edition 24, 35
 - for Plug-In 22
 - in frame 23
 - MWFs in HTML pages 20
- enabling print events 66
- errors 48
 - argument types 48
 - disabling 48
 - enabling 48
 - getting error codes 48
 - handling 47
- event handler
 - browser differences 71
- event handlers
 - ActiveX Control 75
 - applet for Plug-In 83
 - defined 70
 - Java Edition 72
 - Plug-In 72
 - Plug-In and ActiveX Control 75
 - required Java Edition software 25, 39
 - required Plug-In software 34
 - setting up 72
 - writing 78
- event observer
 - defined 70
 - required software, Java Edition 25, 39
 - required software, Plug-In 34
 - set methods 71
 - setting up 72
- events
 - applet for Plug-In 83
 - defined 70
 - detecting busy state 46
 - enabling print events 66
 - handling
 - ActiveX Control 75
 - Plug-In and ActiveX Control 75
 - Plug-In and Java Edition 72
 - page setup 79
 - print 81
 - setting up handlers 72
 - writing handlers for 78
- exceptions. *See* errors

F

- facilities management example 156
- features
 - counting 62
 - map 56
- filtering DWG data 164

G

- generating reports 86
- getLayerSetup 43
- getMap function 32, 33
- getVertices 43

H

- handling busy state and map refresh 43
 - detecting map busy state change 46
 - detecting map refresh 46
- handling errors 47
- handling events
 - ActiveX Control 75
 - Plug-In and ActiveX Control 75
 - Plug-In and Java Edition 72
 - setting up 72
- help
 - for Autodesk MapGuide Viewer API 13
 - for users 11
- HTML pages
 - creating 90, 95, 104, 114, 118, 127

I

- installing client Viewer
 - ActiveX Control 27
 - Java Edition 29
 - Plug-In 28
- intermediate updates 8, 47
- Internet Explorer
 - and observer objects 36
 - embedding a map 22
 - event handling 71
 - event observers in 70
 - JavaScript support in 32, 36
 - map access 32
- invoking radius mode 61

J

- Java 10
 - applet wrapper for Java Edition 38
 - application as wrapper for Java Edition 38
 - peer applet for Java Edition 38

- Java Edition 11, 24, 35
 - displaying the map in 24
 - embedding Java Edition in an HTML page 35
 - embedding maps 35
 - installing on client machine 29
 - Java applet wrapper for 24, 38
 - Java application wrapper for 38
 - Java peer applet for 38
 - map access 35
 - from HTML 35
 - from Java 38
 - from JavaScript/JScript 35
 - platform requirements 9
 - required software 25, 39
 - Viewer differences 39
- Java Server Pages (JSP) 26
- JavaScript 10, 36, 78
 - accessing from an applet 42
 - accessing reports with 100, 108, 122, 133
 - browser support 36
 - Java Edition support 36
 - operating system support 36
- JScript 10, 36, 112
 - browser support 36
 - Java Edition support 36
 - operating system support 36

K

- Key Index Files (KIFs). *See* SDF Component Toolkit
- keys of map features 56

L

- languages 10
- launching reports 87
- layers
 - counting with the API 50
 - DWG 164
 - DWG layer, defined 165
 - DWG map layer, defined 165
 - listing with the API 51
 - map 50
 - adding of 53
 - changing data source properties of 165
 - counting of 50
 - listing of 51
 - visibility 55
 - MapGuide map layer, defined 165
 - visibility 55
- legend, suppressing in printout 64
- linking
 - MWFs to HTML pages 20
 - to map 18, 20
- listing map layers 51
- LiteView. *See* Autodesk MapGuide LiteView

M

MacOS, supported configurations 19

map

- accessing
 - from ActiveX Control 32
 - from Java Edition with Java 38
 - from Plug-In 32
 - from Plug-In with applet 40
- busy state, detecting change in 46
- controlling redraw 47
- displaying 14
 - area of 21
 - in frame 21
- embedding 18
 - for ActiveX Control 22
 - for Java Edition 24
 - for Plug-In 22
- features 50, 56
 - counting 62
 - getting coordinates of 58
 - getting keys 56
- getting the mode of 9
- intermediate updates to 47
- layers 50
 - adding of 53
 - changing data source properties of 165
 - counting of 50
 - DWG properties and 165
 - listing of 51
 - visibility of 55
- linking to 18, 20
- objects 50, 56
- printing
 - customizing title 64
 - suppressing legend 64
 - suppressing North arrow 64
 - suppressing scale bar 64
- redraw operations 8
- refresh detecting 46
- viewing 14

map window file. *See* MWF

Map Window Properties dialog box

- Popup Menu tab 108, 132
- Reports tab 103, 116, 126

MapGuide map layer, defined 165

MapGuideObserver6.class 71, 173

MapGuideObserver6J.class 71

menus, adding items to 108, 132

MGDwgDataSources 8, 165

MGError 47, 48

MGMapObject 50

Microsoft Internet Explorer. *See* Internet Explorer

mode, of map 9

municipal application example 140

MWF 13, 20, 165

N

Netscape Navigator

- embedding a map 22
- event handling 71
- event observers in 70
- JavaScript support in 36
- map access 32

new features 8

- DWG support 8
- Java Edition platform support changes 9
- map mode retrieval 9
- map redraw operations 8, 47
- symbol bitmap support 8

O

object 50

OLE DB

- working with data sources 89, 90, 102, 110, 112, 116, 125

See also DSNs

operating systems supported 9, 18

output, customizing 64

P

page coordinate system (PCS) 67

page elements 67

- adding 68
- positioning for printing 67

page setup events 79

PCS. *See* page coordinate system

Perl 26

platform requirements 9, 18

platform support changes 9

Plug-In 11

- applet for communication 40
- displaying a map in 20
- embedding a map 22
- installing on client machine 28
- linking to a map 20
- required software 34

points, digitizing 140

popup menus, creating 108, 132

printing 64

- customizing 64
- enabling print events 66
- events for 79, 81
- handler functions for print events 78
- priority of 65

programming languages 10

Q

querying data 14
and server-side scripting 86
reports 85

R

radius mode, invoking 61
redlining 10
application example 136
compared to updating SDFs 171
defined 136
refresh 46
reports
about 86
accessing with JavaScript 100, 108, 122, 133
accessing with the Viewer API 100, 108, 122, 133
adding to a map 85, 86
and server-side scripting 85
creating in Autodesk MapGuide Author 92, 103, 116, 126
creating with ASP 109
creating with ColdFusion 89, 142, 146
file types
.asp 109, 112, 116, 125
.cfm 89, 92, 102
generating 86
launching 87
naming for ActiveX Control 87
report script 86
request 87
required software
Java Edition event handling 25, 39
Java Edition platform requirements 9
Plug-In event handling 34

S

scripting languages 10
See also JavaScript, JScript, Visual Basic
SDF Component Toolkit 10
application examples 171
adding points of interest to map 172
converting to an SDF 181
copying an SDF 196
getting information about an SDF 188
development environments 171
security
of Autodesk MapGuide Server 43
of data 43
of Viewer API 43
server-side scripts and applications 85, 86

setting events for event observers 71
Solaris, supported configurations 19
Spatial Data Files (SDFs). *See* SDF Component Toolkit
Spatial Index Files (SIFs). *See* SDF Component Toolkit
stand-alone application 19
supported configurations 18
symbols
adding to a printout 64
bitmap support 8

U

updating data via map 14, 86, 102, 125
URL parameters, reports and 93, 104, 117, 126

V

VBScript 10, 112
defined 70
event handler example 75, 78
setting up event handlers 70, 71, 75
supported configurations 19
using for reports in Active Server Pages 109
See also Visual Basic
Viewer
about 11
ActiveX Control 11, 20
creating an application 15
Java Edition 11, 24
Plug-In 11, 20
types of 11
Viewer API 13
about 11
accessing reports with 100, 108, 122, 133
applications 10, 13, 136
counting map features 62
counting map layers 50
creating an application 15
developing with 17
DWG support 164
events 70
Internet Explorer and 70
Netscape Navigator and 70
listing map layers 51
online help 13
programming languages 10
querying data 14
scripting languages 10
security 43
updating data via map 14
viewing maps 14

- visibility
 - of DWG layers 164
 - of map layers 55
- Visual Basic 10
 - SDF Component Toolkit example 181, 188, 196
 - stand-alone Windows applications 19

W

- Windows
 - supported configurations 19

X

- XML 10

Z

- zooming 61