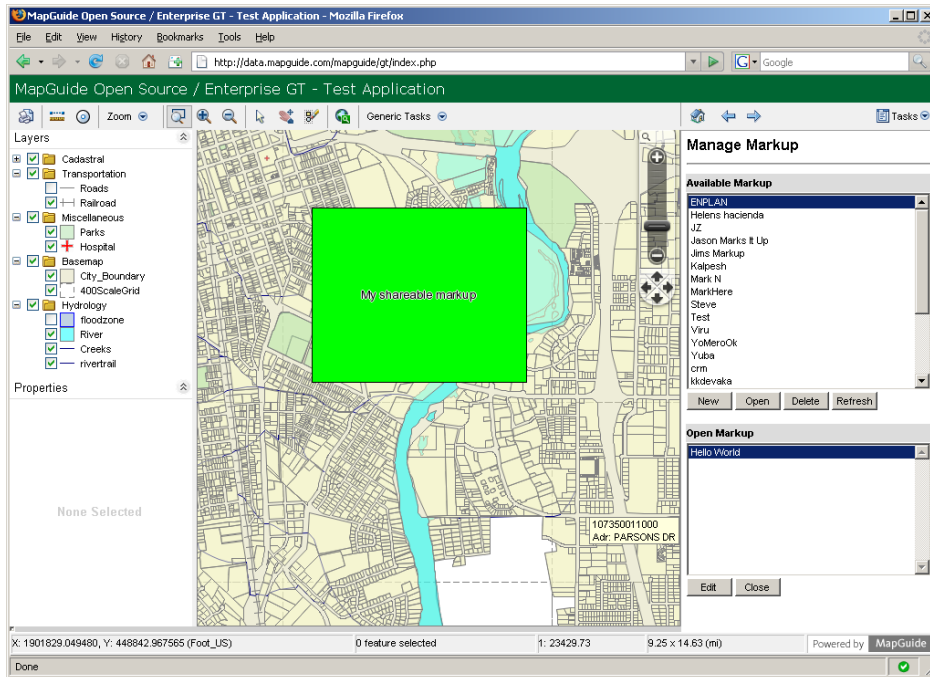


Creating Shareable Markups



Autodesk MapGuide® Enterprise has a powerful API (application programming interface) for creating and managing resources and creating markups or redlines. In addition, the API also exposes FDO Data Access Technology to enable MapGuide-based applications to manipulate data sources directly on the server.

This paper shows how to create markups that are stored on the server and can be shared among multiple users through the MapGuide programming interface. The information in this paper is applicable to both the MapGuide Open Source and Autodesk MapGuide Enterprise versions of the platform.

Contents

Introduction	3
Concepts	4
Enumerating Markup Resources	4
Reading and Writing Markup Resources.....	7
.....	8
Creating the Markup Resources	8
Digitizing Markup Features.....	10
Creating MgGeometry Objects.....	11
Inserting, Updating, and Deleting Markup Features.....	12
Selecting Markup Features Programmatically.....	12
Putting It Together in the Markup Task	13
Resources	15

Introduction

This paper provides an in-depth look at the Autodesk MapGuide Enterprise 2007 APIs and techniques for creating a shareable markup application. In particular it looks at the following:

- Resource Service APIs for enumerating, reading, and writing resources
- Viewer APIs for digitizing geometry, and techniques for passing that to your web application
- Feature Service APIs for creating an SDF data store, selecting features, and updating features
- MgGeometry API and how to create new geometry objects with MgGeometryFactory
- MgSelection API and how to perform selection programmatically

All of the code samples have been taken directly from a sample application that is available on the Autodesk MapGuide Technology Resource Center website (www.autodesk.com/learnmapguide or data.mapguide.com/mapguide/gt/index.php for the direct link to the sample and source code download). The Markup task in the sample application enables the end user to create and share a simple markup (Figure 1).

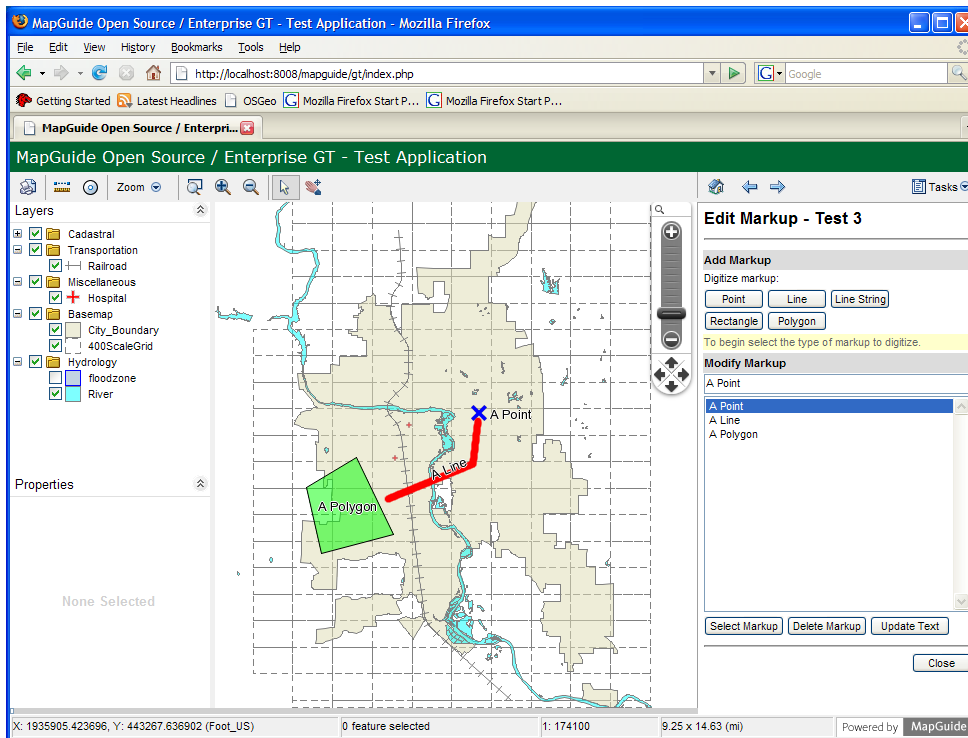


Figure 1: Sample Application Markup Task

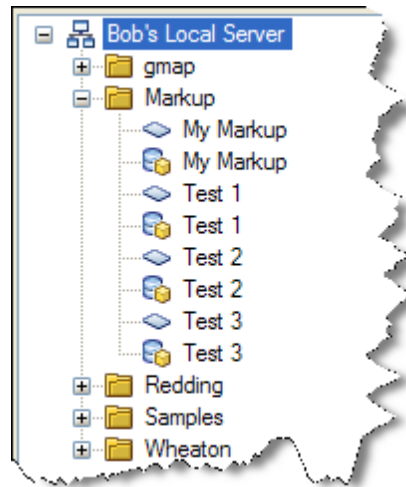
The Manage Markup task enables you to open, delete, or create a new markup layer. Select a markup layer, and click Open to add the markup to the map or click New to create a new markup layer. If you choose New, the New Markup form allows you to name your markup layer and set the point, line, polygon, and label styles. After opening or creating a markup layer, select it at the bottom of the Manage Markup task and click Open

to display the Edit Markup form. Use the tools on this form to digitize new markup features, delete markup features, and update the text value associated with markup features.

All of the Autodesk MapGuide APIs and techniques used to create the Markup application are covered in this paper.

Concepts

Markup within the markup application consists of an SDF data resource that contains the data and a layer definition resource that specifies how the markup should be rendered. To share the markup, these resources are stored in the Library repository in the *Library://Markup* folder. All markup data is stored in the WGS84 projection, so that it can be displayed in any map. When the user creates a new markup layer, the application creates both an SDF data resource and the corresponding layer definition. Opening or closing markup layers simply adds or removes a layer from the current map.



Enumerating Markup Resources

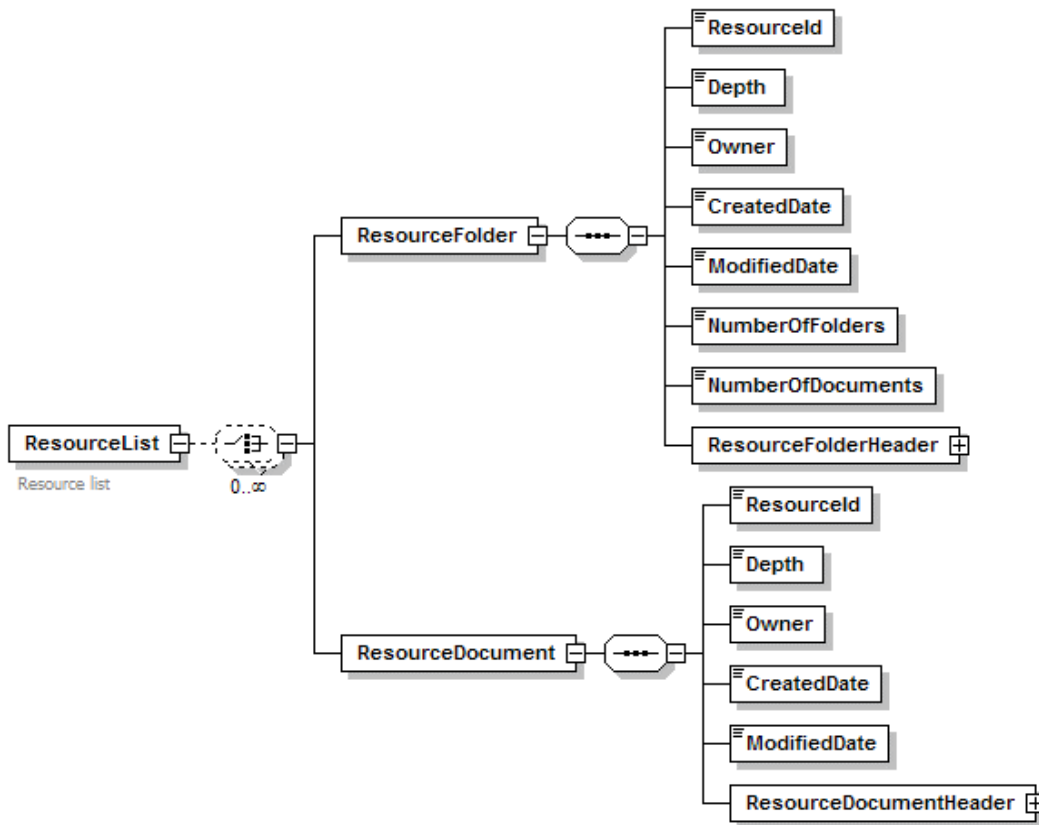
One of the first requirements of the application is to enumerate all of the layer definition resources in the *Library://Markup* folder. You can do so using the `EnumerateResources` method of `MgResourceService`. That method has the following signature:

```
MgByteReader EnumerateResources(resourceId, depth, type);
```

The `resourceId` parameter specifies the root folder in which to enumerate the resources. The `depth` and `type` parameters control how many folders below the root folder are enumerated and the type of resources to be enumerated. A depth of 0 returns only the resources in the specified folder, greater than 0 returns all resources to the specified depth, and -1 returns all resources in all subfolders. The `type` parameter may take on any of the values from the `MgResourceType` enumeration. Putting this method to use, the following code enumerates all resources in the *Library://Markup/* folder.

```
$resourceId = new MgResourceIdentifier("Library://Markup/");
$byteReader = $resourceService->EnumerateResources(
    $resourceId, 1, MgResourceType::LayerDefinition);
$resourceListXML = $byteReader->ToString();
```

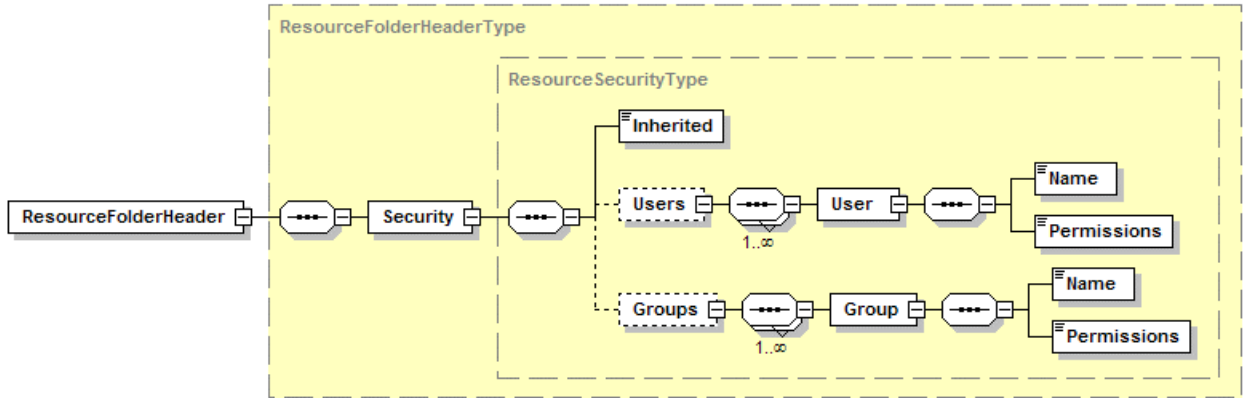
The EnumerateResources method returns a resource list described by an XML document whose structure is defined by the ResourceList schema. The root element ResourceList and its children are described in the following section.



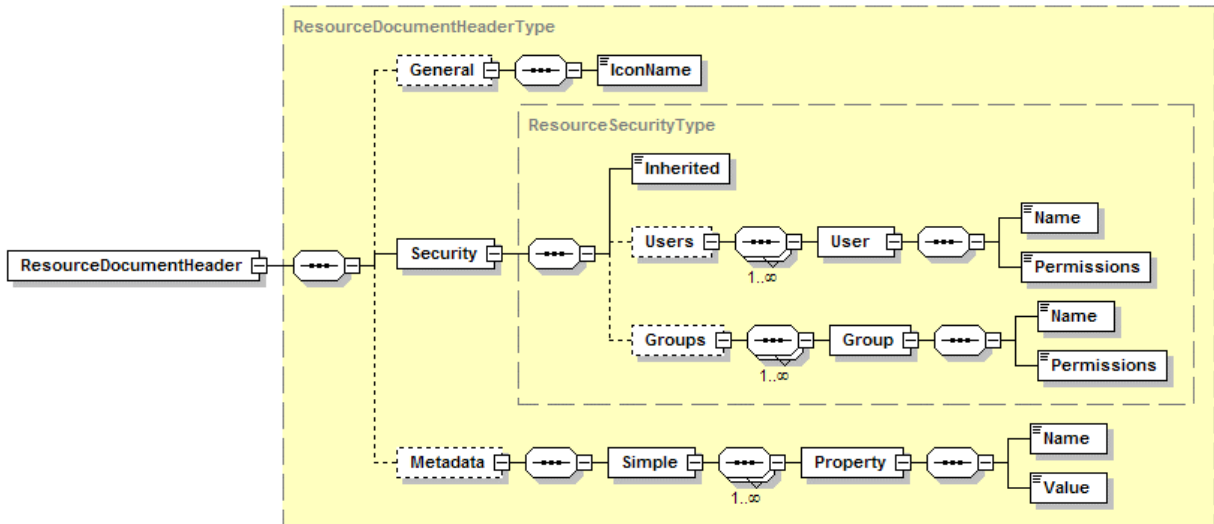
The ResourceList element contains a flat collection of ResourceFolder and ResourceDocument elements. Both elements contain the following elements:

- Resourcecld—Full path and name of the resource
- Depth—Depth of the resource from the root, where the root’s depth is 0
- Owner—UserId of the user who owns the resource
- Created Date—Date the resource was created
- Modified Date—Date the resource was last modified

The ResourceFolder element also contains elements that specify the number of folders and documents contained within it. In addition, the ResourceDocument and ResourceFolder elements both contain a corresponding header element.



The ResourceDocumentHeader element contains the same Security element as the ResourceFolderHeader element. In addition, it can contain General and Metadata elements. The General element contains a single IconName element that specifies the icon to display in Autodesk MapGuide® Studio 2007. The Metadata element contains a collection of name/value pairs, which is commonly used to define Open Geospatial Consortium (OGC) metadata for the resource.



Putting this all together, you can now use the EnumerateResources method to retrieve the layers in the `Library://Markup/` folder and extract all the resource identifiers from the ResourceList XML with the following code:

```

$markup = array();
$resourceID = new
MgResourceIdentifier("//Markup/");

$byteReader = $resourceService->EnumerateResources(
    $resourceID, 1, "LayerDefinition");
$resourceListXML = $byteReader->ToString();

$doc = XmlDocument::LoadXML($resourceListXML);
$nodeList = $doc->getElementsByTagName('Resource');

foreach ($nodeList as $node)
{
    $resourceID = new MgResourceIdentifier($node-
    >nodeValue);
    $markup[$resourceID->ToString()] = $resourceID-
    >GetName();
}

```

Reading and Writing Markup Resources

Reading and writing resource document content is done via the `GetResourceContent` and `SetResource` methods of `MgResourceService`. The signatures of those methods are defined as follows:

```

MgByteReader GetResourceContent(MgResourceIdentifier
resource);

void SetResource(
    MgResourceIdentifier resource,
    MgByteReader content,
    MgByteReader header);

```

The `GetResourceContent` method retrieves the XML content for a specified resource. The `SetResource` method writes the content and header for a specified resource. If the resource does not exist, it is created. If it already exists, it is updated with the new content and header. For `SetResource`, the following applies:

- If content and header are both null, the resource being created must be a folder.
- If header is null and the resource does not yet exist, the security attributes of the parent folder are applied to the resource. If the resource already exists, the header remains unchanged.
- If content is null, the resource must exist and the existing header is replaced with the specified one.

Through these methods, reading a Layer Definition resource from the library can be accomplished as follows:

```

$resId = new
MgResourceIdentifier('Library://MyLayer.LayerDefinition'
);
$byteReader = $resourceService-
>GetResourceContent($resId);
$layerXML = $byteReader->ToString();

```

Subsequently, writing the LayerDefinition resource back to the Library can be accomplished as follows:

```

$resId = new
MgResourceIdentifier('Library://MyLayer.LayerDefinition'
);
$byteSource = new MgByteSource($layerXML,
strlen($layerXML));
$resourceService->SetResource($resId, $byteSource-
>GetReader(), null);

```

Why do I get an access denied error writing to the Library repository?

If you use the preceding code and attempt to write a resource to the Library repository, it may result in an `MgResourcePermission` exception. The most likely cause of this error is that the `UserId` connected to the `MgSite` does not have the `Map Author` role. Any application that needs to write to the Library repository must run in the context of a user that has the `Map Author` role. Also note that an `Anonymous` user cannot be granted the `Map Author` role.

The solution is to create a user using the Autodesk MapGuide 2007 Site Administration application and assign that user the `Map Author` role. Then have your application connect to the `MgSite` and create the session as the newly created user.

Creating the Markup Resources

In the Markup application, creating a new Markup layer requires the creation of an SDF Feature Source Definition and a corresponding Layer Definition resource. You can create a new SDF Feature Source with the `CreateFeatureSource` method of `MgFeatureService`, which has the following signature:

```
CreateFeatureSource(resourceId, featureSourceParams);
```

The `resourceId` parameter specifies the name and path of the feature source to create. The `featureSourceParams` parameter is an object of type `MgFeatureSourceParams` that defines the type of the feature source to create and the parameters required to create it. For Autodesk MapGuide Enterprise 2007, the only concrete subclass of `MgFeatureSourceParams` is `MgCreateSdfParams`. The three key parameters required by `MgCreateSdfParams` are the spatial context name, coordinate system well-known text (WKT), and the schema for the SDF file. The following code fragment creates an SDF feature source for storing a markup:

```

$LL84WKT = ' GEOGCS["LL84", DATUM["WGS_1984", ' .
    ' SPHEROID["WGS
      84", 6378137, 298.25722293287], TOWGS84[0, 0, 0, 0, 0, 0, 0]],
    ' . ' PR I MEM["Greenwi ch", 0], UNI T["Degrees", 1]]';

// Create the ID, Text, and Geometry properties

$idProperty = new MgDataPropertyDefini ti on(' ID' );
$idProperty->SetDataType(MgPropertyType: : I nt32);
$idProperty->SetReadOnl y(true);
$idProperty->SetNul l abl e(fal se);
$idProperty->SetAutoGenerati on(true);

$textProperty = new MgDataPropertyDefini ti on(' Text' );
$textProperty->SetDataType(MgPropertyType: : Stri ng);
$textProperty->SetLength($l ength);

$geometryProperty = new
MgGeometri cPropertyDefini ti on(' Geometry' );
$geometryProperty->SetGeometryTypes(
    MgFeatureGeometri cType: : Poi nt |
    MgFeatureGeometri cType: : Curve |
    MgFeatureGeometri cType: : Surface);
$geometryProperty->SetHasEl evati on(fal se);
$geometryProperty->SetHasMeasure(fal se);
$geometryProperty->SetReadOnl y(fal se);
$geometryProperty->SetSpati al ContextAssoci ati on(' LL84' );

// Create the Markup class

$markupCl ass = new MgCl assDefini ti on();
$markupCl ass->SetName(' Markup' );
$properti es = $markupCl ass->GetProperti es();

$properti es->Add($idProperty);
$properti es->Add($textProperty);
$properti es->Add($geometryProperty);

$markupCl ass->GetI denti tyProperti es() ->Add($idProperty);
$markupCl ass-
>SetDefaul tGeometryPropertyName(' Geometry' );

// Create the Markup Schema

$markupSchema = new MgFeatureSchema();
$markupSchema->SetName(' MarkupSchema' );

$markupSchema->GetCl asses() ->Add($markupCl ass);

// And finally create the SDF Feature Source

$sdfParams = new MgCreateSdfParams(
    ' LL84',
    $LL84WKT,
    $markupSchema);
$featureServi ce->CreateFeatureSource($markupSdfResl d,
    $sdfParams);

```

Once the SDF feature source has been created to store the markup, a Layer Definition resource must be created to define the stylization of the markup features. Doing that using XML APIs such as DOM or even building up the XML using string concatenation can be daunting and error prone. A much better approach is to use Autodesk MapGuide Studio 2007 to create a resource and save it to disk using the File>Save as XML command. You

can then edit the XML file with a text editor and replace element or attribute values that need to be set programmatically with a substitution character. Your application can then load the template file and substitute values in place of the substitution characters. To replace just the resource identifier of the feature source, you can use an XML template for Layer Definition resources that looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<LayerDefinition>
  <VectorLayerDefinition>
    <ResourceId>%s</ResourceId>
    [Content Removed for Clarity]
  </VectorLayerDefinition>
</LayerDefinition>
```

Note the “%s” substitution characters used in the ResourceId element. Now with the following two lines of code, you can have a fully defined Layer Definition resource.

```
$layerDefTemplate =
file_get_contents("templates/layerdeftemplate.xml");
$layerDefXML = sprintf($layerDefTemplate,
$markupSdfResId->ToString());
```

This technique can be applied to snippets of an XML resource document or to entire resource documents. How much you template depends on the requirements of your application. The Markup application uses a single template for the markup Layer Definition resources and substitutes more than 40 element values.

Digitizing Markup Features

Capturing markup feature geometry is accomplished using Autodesk MapGuide Viewer software’s built-in digitizing APIs. The viewer supports six digitizing APIs that allow applications to digitize point, line, rectangle, line string, polygon, and circle geometry. Each digitize operation is based on a callback as follows:

```
function MyFunction()
{
  parent.parent.mapFrame.DigitizePoint(OnDigitizedPoint
());
}

function OnPointDigitized(point)
{
  alert(point.X + ", " + point.Y);
}
```

To pass the digitized geometry to server-side script, you can write the geometry to a string and set a hidden form variable to the value of that string and then submit the form. For example, the following JavaScript code can be used to digitize a line string, convert it to a string, and set a form value to the value of the string.

```

function AddLineString()
{
    parent.parent.mapFrame.DigitizeLineString(OnLineStringDigitized);
}

function OnLineStringDigitized(LineString)
{
    var geomText = LineString.Count;
    for (var i = 0; i < LineString.Count; i++)
    {
        geomText += ", " + LineString.Point(i).X + ", " +
LineString.Point(i).Y;
    }

    var geometryText =
document.getElementById("GEOMTEXT");
    geometryText.value = geomText;
}

```

Note the format of the geomText variable is "Count, V₁X, V₁Y, ..., V_nX, V_nY", where Count is the number of vertices.

Creating MgGeometry Objects

The MgGeometryFactory class is your one-stop shop for creating all of the geometry types supported by MapGuide. It supports everything in OGC simple features as well as circular arcs in line strings and polygons. The following code fragment creates an MgPoint with coordinates 10, 10:

```

$geometryFactory = new MgGeometryFactory();
$coord = $geometryFactory->CreateCoordinateXY(10, 10);
$point = $geometryFactory->CreatePoint($coord);

```

Picking up where the previous section left off, you can create a line string geometry from the text encoded in the OnLineStringDigitized method using the following code fragment:

```

$geometryFactory = new MgGeometryFactory();

$vertices = explode(' ', $this->args['GEOMETRY']);
$count = $vertices[0];

$coords = new MgCoordinateCollection();
for ($i = 0; $i < $count; $i++)
{
    $coord = $geometryFactory->CreateCoordinateXY(
        (double) $vertices[( $i * 2 ) + 1],
        (double) $vertices[( $i * 2 ) + 2]);
    $coords->Add($coord);
}

$lineString = $geometryFactory->CreateLineString($coords);

```

Inserting, Updating, and Deleting Markup Features

Inserting, updating, and deleting feature data is done via a single UpdateFeatures method in MgFeatureService. UpdateFeatures can perform multiple insert, update, and delete operations in a single call. Its signature is as follows:

```
UpdateFeatures(resourceId, commands, useTransaction);
```

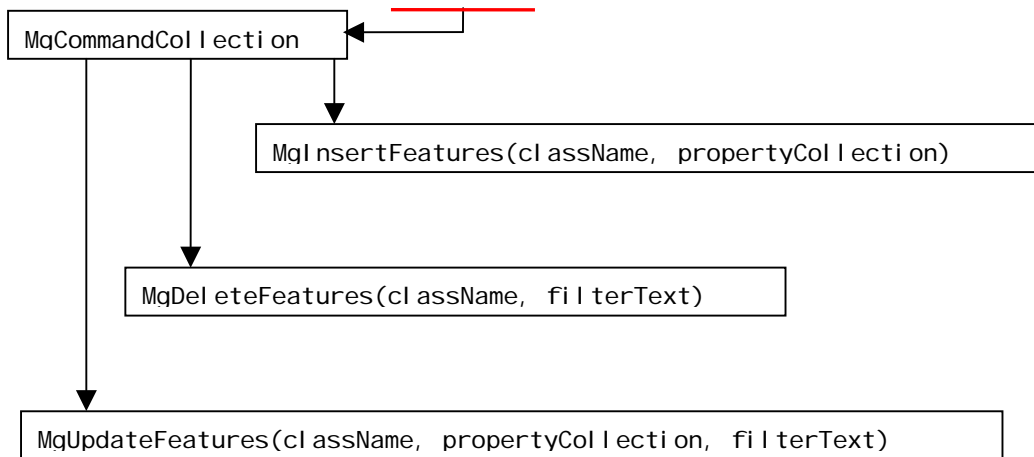


Figure 2: UpdateFeatures Command Collection

The resourceId parameter specifies the feature source the update is to operate upon. The commands parameter specifies a collection of insert, update, and delete operations to perform, as shown in Figure 2. The third parameter, useTransaction, specifies whether the data store should perform all of the operations within the context of a single transaction and commit that transaction when finished.

Selecting Markup Features Programmatically

The MapGuide Viewer supports selection of multiple features across one or more layers. Selection information is passed between client and server using a FeatureSet XML document. The FeatureSet XML schema is depicted in Figure 3.

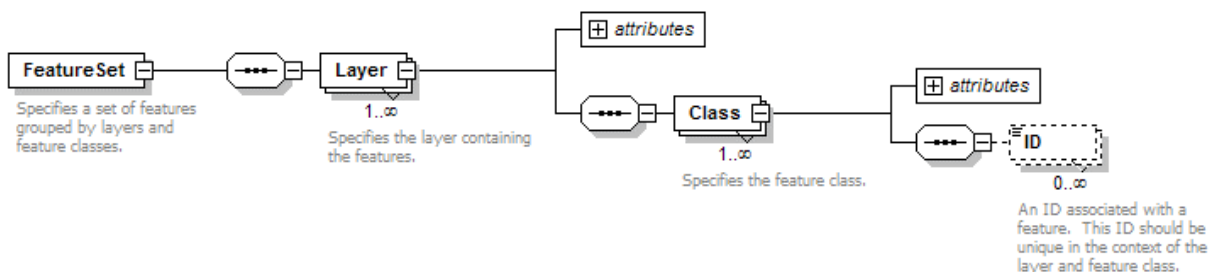


Figure 3: FeatureSet Schema

In the Viewer, the MapFrame contains Get/SetSelectionXML functions, which allow an application to programmatically retrieve or set the selection. On the server side, Autodesk MapGuide Enterprise represents selection using the MgSelection class, which serves two purposes:

- To support programmatic generation of a selection from feature data without manipulating XML
- To format filter statements to easily retrieve feature data for the selection using the Feature Services API

The Markup application allows the user to select a markup feature in a list and select it on the map. In this scenario the MgSelection class is used to generate the FeatureSet XML, which is achieved with the following code fragment:

```
$selection = new MgSelection($map);
$selection->AddFeatureIdInt32($markupLayer,
$markupClass, $markupId);
$featureSetXML = $selection->ToXML();
```

Putting It Together in the Markup Task

Autodesk MapGuide Enterprise 2007 applications typically employ a combination of server-side and client-side logic, where the client-side code is focused on user interaction and the server-side code is focused on geospatial processing. Figure 4 illustrates the typical structure and client/server interaction used by Autodesk MapGuide Enterprise 2007 applications.

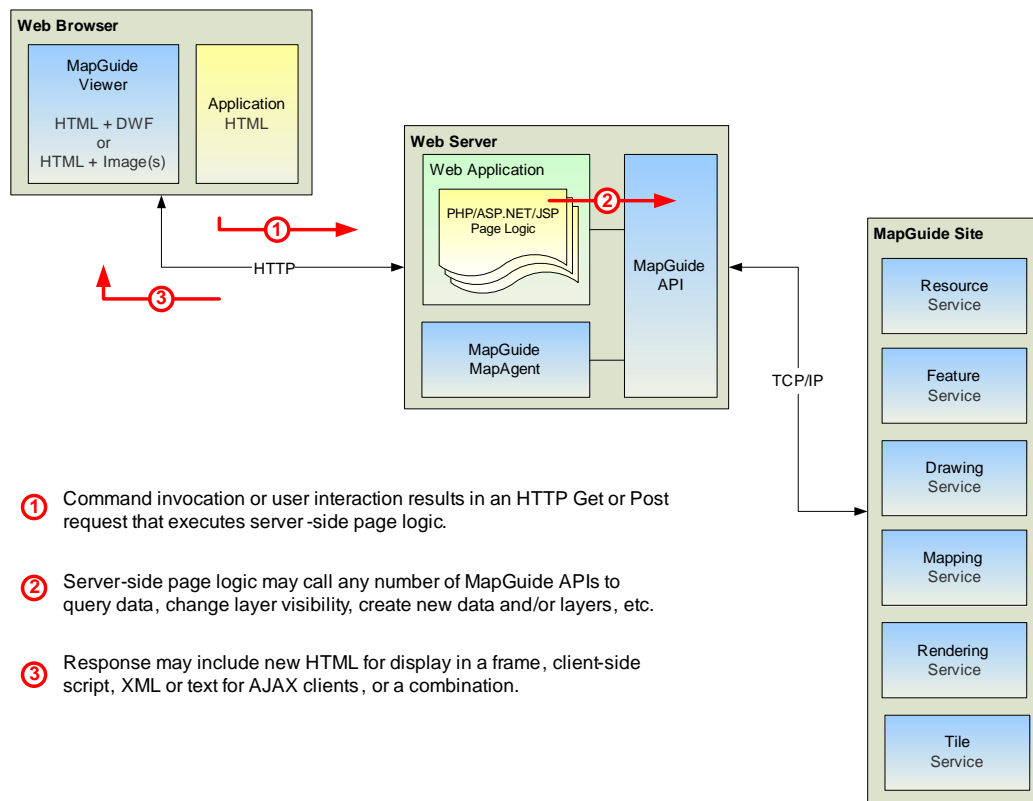


Figure 4: MapGuide Application Interaction

The Markup task is written in PHP and follows this model using standard HTML form techniques for sending post requests to the server-side page logic. The application is made up of the nine files shown in Figure 5.

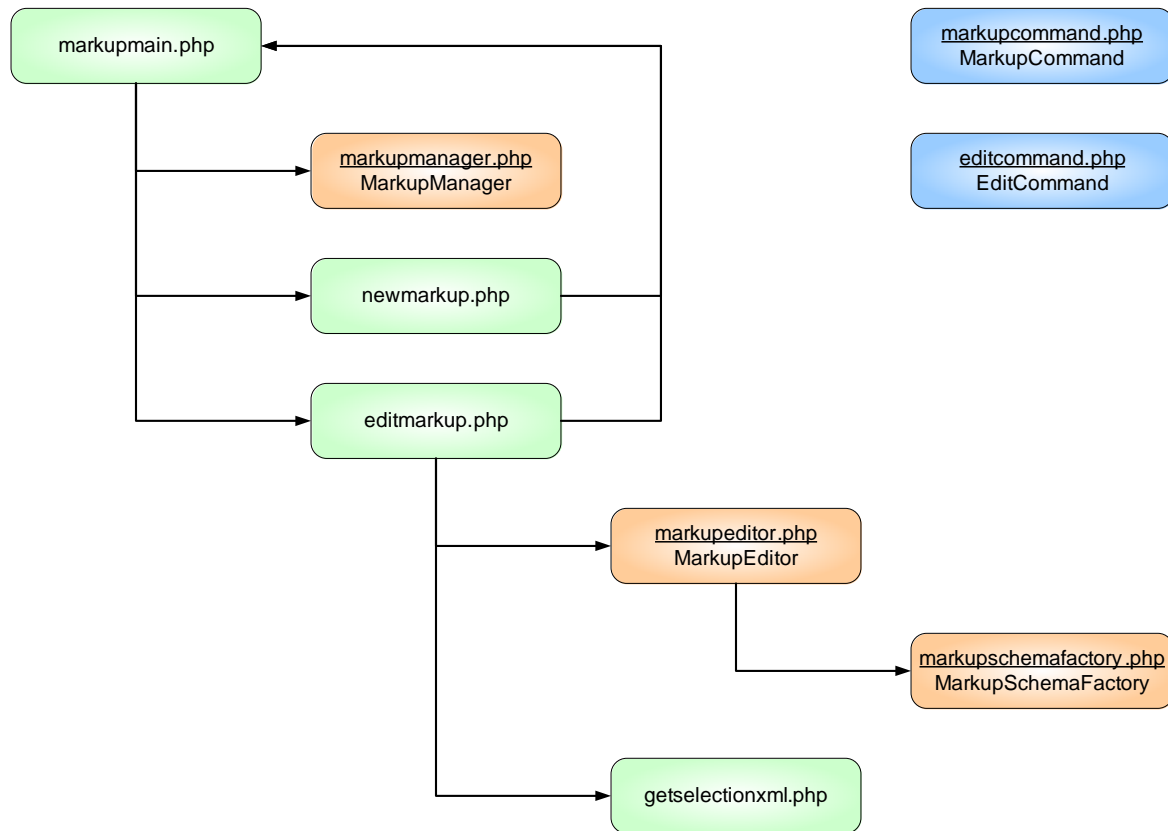


Figure 5: Theme Task Structure

The Theme Task user interface is encapsulated in *markupmain.php*, *newmarkup.php*, and *editmarkup.php*. Form requests from *markupmain.php* come into *markupmanager.php*, *newmarkup.php*, or *editmarkup.php*. The latter two display the corresponding UI panels; the former contains the MarkupManager class, which handles all the server-side processing for *markupmain.php* and *newmarkup.php*. The key operations supported by the MarkupManager class are as follows:

Operation	Description
GetAvailableMarkup	Returns a list of all the LayerDefinition resources in the <i>Library://Markup/</i> folder as an array.
OpenMarkup	Adds the specified markup layer definition to the top of the draw order in the current map.
CloseMarkup	Removes the specified markup layer definition from the current map.
CreateMarkup	Creates the specified markup layer and associated SDF file using the specified stylization parameters.
DeleteMarkup	Deletes the specified markup layer.
GetOpenMarkup	Returns a list of the Markup layers that have been added to the current map.

The editmarkup UI form relies on the MarkupEditor class for all of its server-side processing. The key operations supported by the MarkupEditor class are as follows:

Operation	Description
GetMarkupFeatures	Returns a list of the features that have been added to the markup layer being edited.
AddPoint	Adds a point feature to the markup layer being edited.
AddLineString	Adds a line string feature to the markup layer being edited.
AddPolygon	Adds a polygon feature to the markup layer being edited.
DeleteMarkup	Deletes the specified markup feature from the markup layer being edited.
UpdateMarkup	Updates the text property of the specified markup feature on the markup layer being edited.
GetSelectionXML	Returns the FeatureSet XML document that corresponds to the specified markup feature on the markup layer currently being edited.

The MarkupSchemaFactory class contains utility methods for creating and initializing the schema of the SDF Feature Sources used for storing markup.

Resources

- **Source code and sample application download**—The sample application used in this paper is available online at www.autodesk.com/mapguide under developer samples or the direct link at <http://data.mapguide.com/mapguide/gt/index.php>.
- **Autodesk MapGuide product center**—Get the latest information on all versions of MapGuide including the free data migration tool from the official Autodesk website at www.autodesk.com/mapguide.
- **Autodesk MapGuide discussion group**—Here you will find the MapGuide discussion group, which is an excellent resource for sharing and obtaining information from your peers. Visit www.autodesk.com/discussiongroup-mapguide.
- **Autodesk MapGuide developer discussion group**—The MapGuide developer discussion group is another great resource for development relation issues. Visit www.autodesk.com/discussiongroup-mapguide-developer.
- **MapGuide Open Source website (OSGeo.org)**—This site is the destination for information about MapGuide Open Source. Download the latest release from this site and join the mailing lists at mapguide.osgeo.org.
- **Autodesk® DWF™ Viewer product center**—Download the DWF-based viewer, which can be used with MapGuide Open Source, at www.autodesk.com/dwfviewer.
- **Autodesk DWF developer center**—Find developer tools for the DWF Viewer, including the DWF Viewer API, which can be used with MapGuide Open Source, at www.autodesk.com/developdwf

Autodesk, Autodesk MapGuide, and DWF are registered trademarks or trademarks of Autodesk, Inc., in the USA and/or other countries. OSGeo is a trademark of the Open Source Geospatial Foundation in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product offerings and specifications at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document. © 2007 Autodesk, Inc. All rights reserved.