Table of Contents

WorldWide Telescope Layer Control API (LCAPI)

This document describes how to write a software program to send data to WorldWide Telescope, using the Layer Control API (LCAPI). For a large amount of data the program might be a tool that reads data from a spreadsheet file, and then sends the appropriate fields of the data to WorldWide Telescope in appropriately sized buffers for visualization. For a smaller amount of data the entire file could be loaded in one go, or images or 3D models could be loaded. Time series data would typically be time-and-location dependent event data such as:

- Earthquakes
- Volcanic eruptions
- Disease outbreaks
- More general social or natural events and trends

The time-series system does *not* lend itself to data that varies either its location or has a complex intensity - such as weather systems, or forest fires - though it is possible a limited approach to this sort of data may be useful. The system does lend itself to events that occur over an extended period of time, given the ability to greatly accelerate simulated time, and also events that decay very rapidly (lightning) or quite slowly (diseases) given the ability to control the decay time of the rendered graphic. Although most examples are of time series events on the Earth, the events can be on any of the supported Solar System bodies, or simply the Sky.

There are a number of constraints on the use of the LCAPI that should be understood before commencing with the design of the tool:

- The supported spreadsheet formats are comma or tab delimited files.
- The LCAPI is a feature of the Windows Client version of WorldWide Telescope, not the Web Client.
- The LCAPI provides event, image or model data specifically to the Layer Manager, not to any other component of WorldWide Telescope.

What it can do: In addition to using the layer manager interactively, you can use scripting and programming languages to send commands and data into WWT from your own code driving the visualization of data in WWT. You can write code that can read data from a database, or create it mathematically and send it to WWT to render. In addition almost all of WWT functionality can be controller through the LCAPI. This means you can create control

mechanisms for automated or interactive control of WWT through your own hardware and software. A good example of the use of the LCAPI is the ADS All-Sky Survey as seen in WorldWide Telescope, at adsass.org/wwt/.

Skills Required: You need to be familiar with the programing language or your choice and get the LCAPI documentation and learn how to use HTTP request to send and receive data from WWT. There is also a library called_Narwhal_that provides a .NET wrapper and high level programming features for those who use .NET. Any language that can call HTTP web services can control WWT, and a Python interface has already been implemented, as pyWWT, (see www.jzuhone.com/pywwt).

Design Overview

The Windows client version of WorldWide Telescope listens continuously, and without any setup process, for a certain format of communication on a particular port. The format of this communication is in the form of strings, that themselves are formatted in a specific way. An application may be quite short and only need a **load** call, or perhaps just need the **new** and **update** commands**.** Typically a simple application will go through the following steps, *after* WorldWide Telescope has been started:

1. Establish the IP address of the computer running WorldWide Telescope. If this is the same computer as the application use the getIP utility, if not a utility will have to be written or acquired that establishes the remote IP address.
2. Initialize and open a data file, or files, from a selected source.
3. Initialize the connection with WorldWide Telescope with a **new** or **load** call, which will create a new layer within WorldWide Telescope, and return a layer Id number.
4. If the **new** command is used:
5. Fill a string buffer with data from the opened files.
6. Transmit the data to WorldWide Telescope with an **update** call, using the layer Id number for reference.
7. Repeat this step until all the data has been transmitted.
8. Use the **getprop**, **setprop**, **uisettings** commands and parameters to control the view in WorldWide Telescope from the LCAPI application.
9. Close down.

A more complex application may initialize several layers, or a layer group, within WorldWide Telescope, or may transmit the data under user control - enabling the user to have some choice over which data is rendered, the speed and time of the simulation, and other similar controls.

Development Environment

The samples and examples described in this document have been developed using Microsoft Visual Studio (2008 or 2010) and the .NET framework, and written in C#.

The suggested development approach is first to get the sample application working, then to modify it as required using the code examples for each command. Note that the code examples themselves are not run-able code, but just guidelines on how the calls can be made.

LCAPI Commands

The following commands can be used to control layers from the application. Note that command names are *not* case-sensitive, and that the general parameters can be included along with any other command.

Command	Description
activate	Highlights the selected layer in the layer manager.
delete	Specifies that a layer should be permanently deleted.
getprop	Used to retrieve a value of a single layer property. Remarks include the Table of Properties.
getprops	Used to retrieve all the properties for a specified layer. Includes lists for Spreadsheet, 3d Model, Shapefile and ImageSet.
group	Specifies that a layer group should be added.
layerlist	Returns the structure of the layers and layer group names (in an XML document format) that are currently in the layer manager.
load	Specifies a data file, and and some optional parameters, to apply to a new layer.
mode	Changes the view to one of **Earth**, **Planet**, **Sky**, **Panorama**, **SolarSystem**.
move	Changes the view depending on the supplied parameter.
new	Specifies that a new layer should be created.
setprop	Used to specify a value of a single layer property.
setprops	Used to specify multiple properties for a layer.
state	Requests some details of the current view.
uisettings	Used to change user interface settings, without altering the layer data.
update	Specifies that the data attached to this command should be added to the layer.
version	Returns the version number of the running version of the LCAPI.
general parameters	Parameters that can be applied to any of the commands.

activate

The **activate** layer command will highlight the selected layer in the layer manager.

Remarks

Required Parameters	Description
&id	Specifies the id number of the layer.

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

delete

The **delete** command specifies that a layer should be permanently deleted.

Remarks

All sub-components of the layer will also be deleted.

Required Parameters	Description
&id	Specifies the id number of the layer.

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

Example Code

```
//
// Send a DELETE command
//
WebClient client = new WebClient();
    string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=delete&id={1}", getI
P().ToString(), layerId);
    string response = client.UploadString(url, "");
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(response);
    XmlNode node = doc["LayerApi"];
    string s = node.InnerText;
    //
    // Handle an error situation
    //
    if (s.Contains("Error"))
    {
        throw new Exception(s);
    }
```

getprop

The **getprop** command is used to retrieve a value of a single layer property.

Remarks

Required Parameters	Description
&id	Specifies the id number of the layer.
&propname	Property name, one from the table below.

Return Value

If the call is successful the response will contain the following string:

```
<Status>Success</Status><Layer _propertyName_="_propertyValue_"</Layer></LayerApi>
```

Where *propertyName* is the property name requested, and *propertyValue* is the value returned. For example:

Get properties

```
<LayerApi><Status>Success</Status><Layer AltType="Depth"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer AltColumn="3"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer Astronomical="False"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer BeginRange="1/9/2009 11:44:38 AM"</Layer></La
yerApi>
<LayerApi><Status>Success</Status><Layer EndRange="9/9/2009 11:13:52 PM"</Layer></Laye
rApi>
<LayerApi><Status>Success</Status><Layer Decay="16"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer Name="EarthQuakes2009"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer FadeSpan="00:00:00"</Layer></LayerApi>
<LayerApi><Status>Success</Status><Layer MarkerIndex="-1"</Layer></LayerApi>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid parameter</Status>
```

Example Code

Get altitude units

```
WebClient client = new WebClient();
string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=getprop&id={1}&propname=
{2}", getIP().ToString(), layerId, "AltUnit");
```

response

```
<?xml version="1.0" encoding="utf-8"?><LayerApi><Status>Success</Status><Layer AltUnit=
"Kilometers"</Layer></LayerApi>
```

Table of Properties

The following table lists the properties that can be get or set on a layer (note that the property names and Enum values are case-sensitive). Also refer to the Notes below the table.

Property	Type	Description	Layer Type
AltColumn	Int	Column number containing the altitude/depth data.	Spreadsheet
AltType	Enum	One of: **Depth, Altitude, SeaLevel, Terrain**.	Spreadsheet

AltUnit	Enum	One of: **Meters, Feet, Inches, Miles, Kilometers, AstronomicalUnits, LightYears, Parsecs, MegaParsecs.**	Spreadsheet
Astronomical	Boolean	True if the data is astronomical rather than terrestrial. If this is set to true the declination will be by default the Latitude column, and the right ascension the Longitude column.	All
BeginRange	DateTime	Date and time of the first data entry in a time series, in one the formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	Spreadsheet
CartesianCustomScale	Double	Used to divide the Cartesian co-ordinate values to a custom scale. If this value was set to "0.01" and the scale to Meters, the custom scale would be "centimeters".	Spreadsheet
CartesianScale	Enum	If the **CoordinatesType** is not **Spherical** then the co-ordinate system uses X,Y and Z values. One of: **Meters, Feet, Inches, Miles, Kilometers, AstronomicalUnits, LightYears, Parsecs, MegaParsecs, Custom**.	Spreadsheet
ColorMap	Enum	One of: **Same_For_All, Group_by_Values.**	Spreadsheet
ColorMapColumn	Int	Column number containing the color map data. Columns are numbered from zero, -1 indicates there is no data for this.	Spreadsheet
ColorValue	String	String containing ARGB value of the color, in the format: "ARGBColor:255:255:255:255".	All
		One of: **Spherical, Rectangular, Orbital**. **Spherical** applies when the	

CoordinatesType	Enum	reference frame is a sphere, and the coordinates will be Lat/Lng or RA/Dec. **Rectangular** applies if there are X,Y and Z coordinates. 0,0,0 in this case is the center of the reference frame.	Spreadsheet
Decay	Float	Decay rate of the visualization, in days, in the range 0.00025 to 4096.	Spreadsheet
Enabled	Boolean	True if the layer is enabled.	All
EndDateColumn	Int	Column containing the end date/time data.	Spreadsheet
EndRange	DateTime	Date and time of the last data entry in a time series, in one the formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	Spreadsheet
EndTime	DateTime	Date and time to end the visualizations, and return to the start time in the case of an auto loop. This property is visible in the Lifetime dialog for a layer. Format (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	All
GeometryColumn	Int	Column containing the geometry data.	Spreadsheet
Heading	Double	Heading angle of the model in radians.	3D Model
HyperlinkColumn	Int	Column containing the hyperlink data.	Spreadsheet
HyperlinkFormat	String	Spreadsheet	""
FadeSpan	Time	The time it takes to fade out the graphic. This property is visible in the **Lifetime** dialog for the layer and is shown in days in the dialog. When using the SDK use the format D.HH:MM:SS, so for one day	All

		use **1.00:00:00**.	
FadeType	Enum	One of: **In, Out, Both, None**.	All
FlipV	Boolean	True if the textures for the 3D model should be flipped.	3D Model
LatColumn	Int	Column number for the Latitude data.	Spreadsheet
LngColumn	Int	Column number for the Longitude data.	Spreadsheet
MarkerColumn	Int	Column number for the marker data.	Spreadsheet
MarkerIndex	Int	Index number of marker graphic selected, or -1 if no marker has been selected.	Spreadsheet
MarkerMix	Enum	One of: **Same_For_All, Group_by_Values.**	Spreadsheet
MarkerScale	Enum	One of: **Screen, World**. If this is set to **Screen** the graphic will not change in size as the viewer zooms in and out. If it is set to **World** the graphic (**Pushpin** or **Gaussian**) will scale with the view.	Spreadsheet
Name	String	Name of the layer.	All
NameColumn	Int	Column containing the event name data.	Spreadsheet
Opacity	Float	Opacity of the layer graphics from 0.0 to 1.0.	All
Pitch	Double	Pitch angle of the model in radians.	3d Model
PlotType	Enum	One of: **Gaussian, Point, PushPin**.	Spreadsheet
PointScaleType	Enum	One of: **Linear**, **Power, Log, Constant, StellarMagnitude.**	Spreadsheet
RaUnits	Enum	When the longitude column is being used for RA in astronomical data. One of: **Hours**, **Degrees**.	Spreadsheet
Roll	Double	Roll angle of the model in radians.	3D Model
Scale	Vector3d	Scale of the model in x, y and z dimensions. Format: 1,1,1	3D Model

ScaleFactor	Float	Scale factor to apply to the magnitude of the event.	Spreadsheet
ShowFarSide	Boolean	True if the data points on the invisible side of the body should be shown.	Spreadsheet
SizeColumn	Int	Column containing the size (magnitude) data.	Spreadsheet
Smooth	Boolean	True if the 3D Model should have its normals smoothed.	3D Model
StartDateColumn	Int	Column containing the start date/time data.	Spreadsheet
StartTime	DateTime	Date and time to start the visualizations. This property is visible in the Lifetime dialog for the layer. Formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	All
TimeSeries	Boolean	True if the layer should be treated as time series data.	Spreadsheet
Translate	Vector3d	Translation (movement offset) of the model in x, y and z dimensions, in units of the model size. Format: 1,1,1	3D Model
XAxisColumn	Int	Column number for X data.	Spreadsheet
YAxisColumn	Int	Column number for Y data.	Spreadsheet
ZAxisColumn	Int	Column number for Z data.	Spreadsheet
XAxisReverse	Boolean	True if the direction of the X axis is reversed.	Spreadsheet
YAxisReverse	Boolean	True if the direction of the Y axis is reversed.	Spreadsheet
ZAxisReverse	Boolean	True if the direction of the Z axis is reversed.	Spreadsheet

Notes

- Columns are numbered from zero. A value of -1 for any column entry indicates that there is no data of that type.
- AM and PM in times: The time 1/1/2000 12:00:00 AM is equivalent to midnight on the

previous night - so is one second after 12/31/1999 23:59:59. If the time was set to PM rather than AM then 12:00:00 references noon.

getprops

The **getprops** command is used to retrieve all the properties for a specified layer.

Remarks

Refer to the table of properties listed for the **getprop** command.

Required Parameters	Description
&id	Specifies the id number of the layer.

Return Value

If the call is successful the response string will contain the following string:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

Example Code

The following example gets all the properties from any of the different types of layer. Note that the response string does not include newlines, these have been added for readability.

get all properties

```
 private void buttonGetProperties()
 {
    try
    {
        WebClient client = new WebClient();
        string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=getprops&id={1}"
, getIP().ToString(), layerId);
        string response = client.UploadString(url, "");
        XmlDocument doc = new XmlDocument();

        doc.LoadXml(response);
        XmlNode node = doc["LayerApi"];
        if (node.InnerText.Contains("Success"))
        {
            XmlNode node2 = node["Layer"];
            if (node2 != null)
            {
                string propName;
                string propValue;

                for (int i = 0; i < node2.Attributes.Count; i++)
                {
                    propName = node2.Attributes[i].Name;
                    propValue = node2.Attributes[i].Value;
                    // Do something with the name/value pair
                }
            }
            else
            {
                // No properties returned
            }
        }
        else
        {
           throw new Exception(response);
        }
    }
    catch (Exception ex)
    {
        // Handle exception
    }
}
```

Spreadsheet

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
<Status>Success</Status>
<Layer
   Class=\"SpreadSheetLayer\"
   TimeSeries=\"True\"
   BeginRange=\"1/9/2009 11:44:38 AM\"
   EndRange=\"9/9/2009 11:13:52 PM\"
   Decay=\"16\"
   CoordinatesType=\"Spherical\"
   LatColumn=\"1\"
   LngColumn=\"2\"
   GeometryColumn=\"-1\"
   XAxisColumn=\"-1\"
   YAxisColumn=\"-1\"
   ZAxisColumn=\"-1\"
   XAxisReverse=\"False\"
   YAxisReverse=\"False\"
   ZAxisReverse=\"False\"
   AltType=\"Depth\"
   RaUnits=\"Hours\"
   MarkerMix=\"Same_For_All\"
   MarkerColumn=\"-1\"
   ColorMapColumn=\"-1\"
   PlotType=\"Gaussian\"
   MarkerIndex=\"-1\"
   ShowFarSide=\"False\"
   MarkerScale=\"World\"
   AltUnit=\"Kilometers\"
   CartesianScale=\"Meters\"
   CartesianCustomScale=\"1\"
   AltColumn=\"3\"
   StartDateColumn=\"0\"
   EndDateColumn=\"-1\"
   SizeColumn=\"4\"
   NameColumn=\"0\"
   HyperlinkFormat=\"\"
   HyperlinkColumn=\"-1\"
   ScaleFactor=\"1\"
   PointScaleType=\"Power\"
   Opacity=\"1\"
   StartTime=\"1/1/0001 12:00:00 AM\"
   EndTime=\"12/31/9999 11:59:59 PM\"
   FadeSpan=\"00:00:00\"
   FadeType=\"None\"
   Name=\"EarthQuakes2009\"
   ColorValue=\"ARGBColor:255:255:0:0\"
   Enabled=\"True\"
   Astronomical=\"False\" />
</LayerApi>
```

3D Model

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
<Status>Success</Status>
<Layer
   Class=\"Object3dLayer\"
   FlipV=\"False\"
   Smooth=\"True\"
   Heading=\"0\"
   Pitch=\"0\"
   Roll=\"0\"
   Scale=\"1, 1, 1\"
   Translate=\"0, 0, 0\"
   Opacity=\"1\"
   StartTime=\"1/1/0001 12:00:00 AM\"
   EndTime=\"12/31/9999 11:59:59 PM\"
   FadeSpan=\"00:00:00\"
   FadeType=\"None\"
   Name=\"aurora\"
   ColorValue=\"ARGBColor:255:255:0:0\"
   Enabled=\"True\"
   Astronomical=\"False\" />
</LayerApi>
```

Shapefile

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
<Status>Success</Status>
<Layer
   Class=\"ShapeFileRenderer\"
   Opacity=\"1\"
   StartTime=\"1/1/0001 12:00:00 AM\"
   EndTime=\"12/31/9999 11:59:59 PM\"
   FadeSpan=\"00:00:00\"
   FadeType=\"None\"
   Name=\"state_bounds\"
   ColorValue=\"ARGBColor:255:255:0:0\"
   Enabled=\"True\"
   Astronomical=\"False\" />
</LayerApi>
```

ImageSet

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
<Status>Success</Status>
<Layer
   Class=\"ImageSetLayer\"
   Opacity=\"1\"
   StartTime=\"1/1/0001 12:00:00 AM\"
   EndTime=\"12/31/9999 11:59:59 PM\"
   FadeSpan=\"00:00:00\"
   FadeType=\"None\"
   Name=\"Bathymetry\"
   ColorValue=\"ARGBColor:255:255:0:0\"
   Enabled=\"True\"
   Astronomical=\"False\" />
</LayerApi>
```

group

The **group** command specifies that a layer group should be added.

Remarks

Layer groups are just an organizational aid when using the layer manager. The user will be able to collapse and expand groups in the Layer Manager, and have groups that are sub-sets of other groups.

Required Parameters	Description
&name	A unique name for the layer group.
&frame	The reference frame of the group. This can be a layer group created with the **group** command, or one of: **Earth, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, Io, Ganymede, Callisto, Europa, Sun, ISS.**

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

Example Code

```
  private void createLayerGroup(string frame, string name)
  {
      try
      {
          WebClient client = new WebClient();
          string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=group&frame={1
}&name={2}", getIP().ToString(), frame, name);
          string response = client.UploadString(url, "");
          XmlDocument doc = new XmlDocument();
          doc.LoadXml(response);
          XmlNode node = doc["LayerApi"];
          string s = node.InnerText;

          //
          // Handle an error situation
          //
          if (s.Contains("Error"))
          {
              throw new Exception(s);
          }
      }
      catch (Exception ex)
      {

      }
  }
```

layerlist

The **layerlist** command returns the structure of the layers and layer group names (in an XML document format) that are currently in the layer manager.

Remarks

Optional Parameter	Description	Default Value
&layersonly	True indicates that only layers, and not reference frames or group names, should be returned.	False

Return Value

If the call is successful a string will be returned that is an XML document. Refer to the **Example Code** for the format and contents of the document.

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

Example Code

The following code will result in the response given below, though note that one layer (EarthQuakes2009) and one group (New Mercury Group) have been added to the layer manager prior to the **layerlist** call.

layerlist

```
//
// Recursive scanning of child nodes
//
private void scanNode(XmlNode node)
{
   int i,n;
   //
   // First loop through the attributes of the node
   //

   for (i = 0; i < node.Attributes.Count; i++)
   {
      // Do something with the node.Attributes[i].Name and node.Attributes[i].Value
   }

   //
   // Next recurse the node tree
   //
   for (n = 0; n < node.ChildNodes.Count; n++)
   {
      scanNode(node.ChildNodes[n])
   }
}
//
// Get the current list of Reference Frames, groups and layers
//
private void getLayerlist(string layersonly)
{
   try
   {
     WebClient client = new WebClient();
     string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=layerlist&layersonl
```

```
y={1}", getIP().ToString(), layersonly);
     string response = client.UploadString(url, "");
     XmlDocument doc = new XmlDocument();

      doc.LoadXml(response);
      XmlNode node = doc["LayerApi"];
      if (node.InnerText.Contains("Success"))
      {

         XmlNode node2 = node["LayerList"];

         if (node2 != null)
         {
            // Top level items returned = node2.ChildNodes.Count
            scanNode(node2);
         }
         else
         {
            // No items returned
         }
      }
      else
      {
         throw new Exception(response);
      }
   }
   catch (Exception ex)
   {
      // Handle exception
   }
}
```

response (layersonly = "false")

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
<Status>Success</Status>
<LayerList>
<ReferenceFrame Name=\"Sun\" Enabled=\"True\">
<ReferenceFrame Name=\"Mercury\" Enabled=\"True\">
    <LayerGroup Name=\"New Mercury Group\" Enabled=\"True\" />
</ReferenceFrame>
<ReferenceFrame Name=\"Venus\" Enabled=\"True\" />
<ReferenceFrame Name=\"Earth\" Enabled=\"True\">
    <Layer Name=\"EarthQuakes2009\" ID=\"712cdef2-907f-4fc5-ae2c-c57488be4517\" Type="
SpreadSheetLayer" Enabled=\"True\" />
    <ReferenceFrame Name=\"Moon\" Enabled=\"True\" />
    <ReferenceFrame Name=\"ISS\" Enabled=\"True\" />
</ReferenceFrame>
<ReferenceFrame Name=\"Mars\" Enabled=\"True\" />
<ReferenceFrame Name=\"Jupiter\" Enabled=\"True\">
    <ReferenceFrame Name=\"Io\" Enabled=\"True\" />
    <ReferenceFrame Name=\"Europa\" Enabled=\"True\" />
    <ReferenceFrame Name=\"Ganymede\" Enabled=\"True\" />
        <ReferenceFrame Name=\"Callisto\" Enabled=\"True\" />
</ReferenceFrame>
<ReferenceFrame Name=\"Saturn\" Enabled=\"True\" />
<ReferenceFrame Name=\"Uranus\" Enabled=\"True\" />
<ReferenceFrame Name=\"Neptune\" Enabled=\"True\" />
<ReferenceFrame Name=\"Pluto\" Enabled=\"True\" />
</ReferenceFrame>
</LayerList>
</LayerApi>
```

response (layersonly = "true")

```
<?xml version='1.0' encoding='UTF-8'?>
<LayerApi>
  <Status>Success</Status>
  <LayerList>
    <Layer Name=\"EarthQuakes2009\" ID=\"712cdef2-907f-4fc5-ae2c-c57488be4517\" Type="
SpreadSheetLayer"  Enabled=\"True\" />
  </LayerList>
</LayerApi>
```

load

The **load** command specifies a data file, and and some optional parameters, to apply to a new layer.

Remarks

Files that can be loaded using this commend include spreadsheet data (comma or tab delimited), shape files (.shp), 3D model files (.3ds), and WTML files containing **ImageSet** references. In the latter case the first image set found in the file will be loaded. Refer to the WorldWide Telescope Data Files Reference for more details on WTML files.

If a spreadsheet of data is loaded, the **load** command will initiate an import wizard of WorldWide Telescope, in order to select the appropriate columns, timing data, and so on. An import wizard is not automatically invoked by the other data file types.

Note that parameter names are case-sensitive.

Required Parameters	Description
&frame	The reference frame of the group. This can be an existing layer group name, or one of: **Earth, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, Io, Ganymede, Callisto, Europa, Sun, ISS.**
&filename	The full path of the file to load.

Optional Parameters	Description	Default Value
&name	A friendly name for the layer.	"New Layer"
&color	ARBG hex value of the color to be used when rendering the events of the layer.	FFFFFFFF (white)
&startdate	With time series data, the date and time to start the visualization for this layer. This could for example be slightly earlier than the date of the first event in the actual data. For example: `"1/1/2000 12:30:30 AM"`. Formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	The System minimum date value
&enddate	With time series data, the date and time to end the visualization for this layer.	The System maximum date value
&fadetype	Fades the data visualization. One of: **In**, **Out**, **Both** or **None**.	None
&faderange	Fade time in days.	Zero

Return Value

The following string will be included in the response string:

```
<LayerApi><NewLayerID>_nnnn_</NewLayerID></LayerApi>
```

Where *nnnn* is the layer id, which will be a GUID in string format.

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Could not Load Layer</Status>
```

Example Code

Note that the example code uses **datetime**, one of the general parameters, to initiate the visualization of the data immediately.

```
private void loadDatafile(string name, string frame, string filename, string datetime)
{
    try
    {
        WebClient client = new WebClient();
        string url = string.Format("[http://{0}:5050/layerApi.aspx?cmd=load&frame={1}&
filename={2}&name={3}&datetime={4}](http://{0}:5050/layerApi.aspx?cmd=load&frame={1}&f
ilename={2}&name={3}&datetime={4})",
getIP().ToString(), frame, filename, name, datetime);
        string response = client.UploadString(url, "");
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(response);
        XmlNode node = doc["LayerApi"];
        XmlNode child = node.ChildNodes[0];
        layerId = child.InnerText;
        string s = node.InnerText;

        //
        // Handle an error situation
        //
        if (s.Contains("Error"))
        {
            throw new Exception(s);
        }
    }
    catch (Exception ex)
    {

    }
}
```

mode

The **mode** command changes the view to one of **Earth**, **Planet**, **Sky**, **Panorama**, **SolarSystem**.

Remarks

This command does not take any parameters.

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

Example code

Change mode

```
WebClient client = new WebClient();
string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=mode&lookat={1}", getIP(
).ToString(), "SolarSystem");
string response = client.UploadString(url, "");
```

move

The **move** command changes the view depending on the supplied parameter.

Remarks

One move parameter must be supplied, **&move=*nnnn***, where *nnnn* is one of the following:

Required Parameters	Description
ZoomIn	Zoom in on the current view.
ZoomOut	Zoom out of the current view.
Up	Move the current view up.
Down	Move the current view down.
Left	Move the current view left.
Right	Move the current view right.
Clockwise	Rotate the view clockwise 0.2 of one radian.
CounterClockwise	Rotate the view counterclockwise 0.2 of one radian.
TiltUp	Angle the view up 0.2 of one radian.
TiltDown	Angle the view down 0.2 of one radian.
Finder	Currently unimplemented.

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

The following code will be included in the response if the move parameter is invalid:

```
<Status>Error - Invalid parameter</Status>
```

Example code

```
WebClient client = new WebClient();
string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=move&move={1}", getIP().
ToString(), "ZoomIn");
string response = client.UploadString(url, "");
```

new

The **new** command specifies that a new layer should be created.

Remarks

The **new** command will request that an entirely new layer be created, with the following parameters (note that the parameter names are case-sensitive):

Required Parameters	Description
&frame	The reference frame of the layer. This can be one of: **Earth, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, Io, Ganymede, Callisto, Europa, Sun, ISS.**

Optional Parameters	Description	Default Value
&name	A friendly name for the layer.	"New Layer"
&color	ARBG hex value of the color to be used when rendering the events of the layer.	FFFFFFFF (white)
&startdate	With time series data, the date and time to start the visualization for this layer. This could for example be slightly earlier than the date of the first event in the actual data. For example: `html"1/1/2000 12:30:30 AM"`. Formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"	The System minimum date value
&enddate	With time series data, the date and time to end the visualization for this layer.	The System maximum date value
&fadetype	Fades the data visualization. One of: **In**, **Out**, **Both** or **None**.	None
&faderange	Fade time in days.	Zero

The **new** command is embedded in a formatted string (see the **Example Code)**. This formatted string forms the first of two parameters to the **WebClient UploadString** method.

The second string parameter to **UploadString** should contain a comma or tab delimited string containing the heading names of the columns in the layer. The supported column names are listed in the following table. The names of the columns in the data file must *contain* the names listed here (so **"Latitude"** is acceptable for **"Lat"**, or "**dMag**" acceptable for "**Mag**", for example).

Column Heading	Description
Time	UTC time, for example: `html"1/1/2000 12:02:46 AM"`
Lat	Latitude in decimal degrees.
Lon	Longitude in decimal degrees.
RA	Right ascension in decimal degrees.
Dec	Declination in decimal degrees.
Depth	Depth in kilometers.
Altitude	Altitude in meters.
Mag	Magnitude as a floating point number.
X	X co-ordinate, if a Rectangular co-ordinate system is being used.
Y	Y co-ordinate.
Z	Z co-ordinate.

Return Value

The following string will be included in the response string:

```
<LayerApi><NewLayerID>_nnnn_</NewLayerID></LayerApi>
```

Where *nnnn* is the layer id number, which will be a GUID in string format. If a GUID is not received, then the call was not successful.

Example Code

Note that the example code uses **datetime**, one of the general parameters, to initiate the visualization of the data immediately. Required Parameters

```
    //
    // Send a NEW command, extracting info from the data file and the UI
    //
    WebClient client = new WebClient();
    string datetime =  ```html"1/1/2009 12:00:00 AM"```;
    string name = ```html"Earthquakes 2009"```;
    string rate = ```html"10000"```;
    string frame = ```html"Earth"```;
    string color = ```html"FFFF0000"```;

    string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=new&datetime={1}&time
rate={2}&name={3}&frame={4}&color={5}",  getIP().ToString(), datetime, rate, name, fra
me, color);

    // field string below is delimited by tabs, not spaces
    string response = client.UploadString(url, "TIME    LAT    LON    DEPTH    MAG");
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(response);
    XmlNode node = doc["LayerApi"];
    XmlNode child = node.ChildNodes[0];
    layerId = child.InnerText;

    //
    // Handle an error situation
    //
    if (layerId.Length != 36)
        throw new Exception("Invalid Layer Id received");

    //
```

For an example of the use of this command in the sample application, see initWWTLayer.

setprop

The **setprop** command is used to specify a value for a single layer property.

Remarks

Required Parameters	Description
&id	Specifies the id number of the layer.
&propname	Property name. Refer to the table of properties listed for the **getprop** command.
&propvalue	Property value in string form.

Return Value

If the call is successful, then the response will contain the string:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid parameter</Status>
```

Example Code

Set Opacity property

```
 WebClient client = new WebClient();
 string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=setprop&id={1}&propname
={2}&propvalue={3}", getIP().ToString(), layerId,"Opacity", "0.5");
 string response = client.UploadString(url, "");
```

setprops

The **setprops** command is used to specify multiple properties for a layer.

Remarks

Required Parameters	Description
&id	Specifies the id number of the layer.

Note that the second string parameter to the **UploadString** method contains the xml encoding of the properties. The string should be a correctly formatted XML file, with **xml**, **LayerApi** and **Layer** entries. Each attribute to the **Layer** entry should be a property name/property value pair.

This method can be used to set a single property, though the **setprop** command is specifically designed for this.

Refer to the table of properties listed for the **getprop** command.

Return Value

If the call is successful, then the response will contain the string:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid parameter</Status>
```

Example Code

Set BeginRange and Decay properties

```
string propertyXML = "<?xml version='1.0' encoding='UTF-8'?><LayerApi><Layer BeginRang
e=\"1/9/2008 11:44:38 AM\" Decay=\"5.5\" /> </LayerApi>";

WebClient client = new WebClient();
string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=setprops&id={1}", getIP(
).ToString(), layerId);
string response = client.UploadString(url, propertyXML);
```

state

The **state** command requests some details of the current view.

Remarks

This command does not take any parameters. The details returned vary slightly depending on the view mode, and include the current view mode, latitude (or declination) in decimal degrees, longitude (or right ascension) in decimal degrees, zoom factor (0 to 360), angle in radians, rotation in radians, current UTC time, time rate, reference frame, a view token, and zoom text.

Return Value

If the call is successful the following string will be returned:

```
<LayerApi><Status>Success</Status><ViewState> [state information](#State_information)
</ViewState></LayerApi>
```

Example code

Get state

```
private void getState()
{
   try
   {
       WebClient client = new WebClient();
       string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=state", getIP().T
oString());
       string response = client.UploadString(url, "");
       XmlDocument doc = new XmlDocument();
       doc.LoadXml(response);
       XmlNode node = doc["LayerApi"];
       if (node.InnerText.Contains("Success"))
       {
           XmlNode node2 = node["ViewState"];

           if (node2 != null)
           {
               string propName;
               string propValue;

               for (int i = 0; i < node2.Attributes.Count; i++)
               {
                   propName = node2.Attributes[i].Name;
                   propValue =  node2.Attributes[i].Value;
                   // Do something with the name value pair
               }
           }
           else
           {
              // Success message was a false positive, no state was returned
           }
       }
       else
       {
           throw new Exception(response);
       }
   }
   catch (Exception ex)
   {
      // Handle the exception
   }
}
```

State information for each of the five modes:

```
<ViewState
   lookat=\"Earth\"
   lat=\"86.1800140779305\"
   lng=\"4.56191579793961\"
   zoom=\"30.9237645312\"
   angle=\"-1.52\"
```

```
   rotation=\"0\"
   time=\"2/23/2011 7:03:31 PM\"
   timerate=\"1\"
   ReferenceFrame=\"Earth\"
   ViewToken=\"SD8834DFA\"
   zoomText=\"5085 km\">
</ViewState>

<ViewState
   lookat=\"Planet\"
   lat=\"86.1800140779305\"
   lng=\"4.58610934632668\"
   zoom=\"360\"
   angle=\"-1.52\"
   rotation=\"0\"
   time=\"2/23/2011 7:04:19 PM\"
   timerate=\"1\"
   ReferenceFrame=\"Mars\"
   ViewToken=\"SD8834DFA\"
   zoomText=\"59200 km\">
</ViewState>

<ViewState
   lookat=\"Sky\"
   ra=\"10.2626615902966\"
   dec=\"46.1799337621283\"
   zoom=\"30.9237645312\"
   rotation=\"0\"
   time=\"2/23/2011 7:04:51 PM\"
   timerate=\"1\"
   ReferenceFrame=\"Space\"
   ViewToken=\"SD8834DFA\"
   zoomText=\"05:09:14\">
</ViewState>

<ViewState
   lookat=\"Panorama\"
   lat=\"-3.07957843596986\"
   lng=\"-39.9196238171819\"
   zoom=\"117.9648\"
   angle=\"-1.52\"
   rotation=\"0\"
   time=\"2/23/2011 7:05:35 PM\"
   timerate=\"1\"
   ReferenceFrame=\"Panorama\"
   ViewToken=\"SD8834DFA\"
   zoomText=\"19:39:39\">
</ViewState>

<ViewState
   lookat=\"SolarSystem\"
   lat=\"2.25294244185011\"
   lng=\"262.382989924732\"
```

```
    zoom=\"0.000578857064151695\"
    angle=\"-1.52\"
    rotation=\"0\"
    time=\"2/23/2011 7:06:13 PM\"
    timerate=\"1\"
    ReferenceFrame=\"Sun\"
    ViewToken=\"SD8834DFA\"
    zoomText=\"38636 km\">
</ViewState>
```

uisettings

The **uisettings** command is used to change user interface settings, without altering the layer data. Note that the spelling errors in the names of the properties must be matched.

Setting Name	Setting Type
AutoHideContext	True or False
AutoHideTabs	True or False
AutoRepeatTour	True or False
AutoRepeatTourAll	True or False
ConstellationBoundryColor	RGB color (default, "MidnightBlue")
ConstellationFigureColor	RGB color (default, "DarkRed")
ConstellationFiguresFile	String (a filename)
ConstellationSelectionColor	RGB color (default, "DarkGoldenrod")
ContextSearchFilter	String (default "Unfiltered")
DomeTilt	Double (default "0")
DomeTypeIndex	Integer (default, "0")
DomeView	True or False
EclipticColor	RGB color (default, "CornflowerBlue ")
FollowMouseOnZoom	True or False
FovCamera	Integer (default, "0")
FovColor	RGB color (default, "white")
FovEyepiece	Integer (default, "0")
FovTelescope	Integer (default, "0")
FullScreenTours	True or False
GridColor	RGB color (default, "64, 64, 64")

ImageQuality	Integer: 0 to 100 (default, 100)
LargeDomeTextures	True or False
LastLookAtMode	Integer (default, "2")
LineSmoothing	True or False
ListenMode	True or False
LocalHorizonMode	True or False
LocationAltitude	Double (default, "100")
LocationLat	Double (default, "47.64222")
LocationLng	Double (default, "-122.142")
LocationName	String (default, "Microsoft Research Building 99")
MasterController	True or False
ShowClouds	True or False
ShowConstellationBoundries	True or False
ShowConstellationFigures	True or False
ShowConstellationNames	True or False
ShowConstellationSelection	True or False
ShowCrosshairs	True or False
ShowDatasetNames	True or False
ShowEarthSky	True or False
ShowEcliptic	True or False
ShowElevationModel	True or False
ShowFieldOfView	True or False
ShowGrid	True or False
ShowSolarSystem	True or False
ShowTouchControls	True or False
ShowUTCTime	True or False
SolarSystemCosmos	True or False
SolarSystemLighting	True or False
SolarSystemMilkyWay	True or False
SolarSystemMinorOrbits	True or False
SolarSystemMinorPlanets	True or False
SolarSystemMultiRes	True or False

SolarSystemOrbitColor	RGB color (default, "64, 64, 64")
SolarSystemOrbits	True or False
SolarSystemOverlays	True or False
SolarSystemPlanets	True or False
SolarSystemScale	True or False
SolarSystemStars	True or False
StartUpLookAt	Integer (default, "4")

Remarks

This command does not perform any action other than that of changing of the user interface settings.

Return Value

If the call is successful, then the response will contain the string:

```
<Status>Success</Status>
```

Example Code

Fly to a location

```
//
// Sets the property "ShowConstellationFigures" to true.

WebClient client = new WebClient();
string  url = string.Format("http://{0}:5050/layerApi.aspx?cmd=uisettings&ShowConstell
ationFigures=True", getIP().ToString());
string response = client.UploadString(url, "");
```

update

The **update** command specifies that the data attached to this command should be added to the layer.

Remarks

The **update** command will request that data for a layer be updated, with the following parameters (note that the parameter names are case-sensitive):

Required Parameters	Description
&id	Specifies the id number of the layer.

Optional Parameters	Description	Default Value
&hasheader	Set to true if the data has a header row. The header should be the first row of the data.	False
&name	A friendly name to rename the layer.	No change
&nopurge	The sending of an update command will delete events that occur before the start time of any events in the update and that have already decayed. Set this flag to **true** if the event data should not be deleted.	False
&purgeall	Purge (delete) all events.	False
&show	Set to **true** to show the layer, **false** to hide it. True and false are not case-sensitive.	True

The second string parameter to **UploadString** should contain a comma or tab delimited string containing the data in a form that matches the heading names of the columns in the layer provided in the **new** command.

Return Value

The following string will be included in the response if the call is successful:

```
<Status>Success</Status>
```

If the call is not successful the following string may be included in the response, or other errors may be returned:

```
<Status>Error - Invalid layer ID</Status>
```

Example Code

```
  //
  // Send an UPDATE command
  //
  WebClient client = new WebClient();
  string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=update&id={1}", getIP(
).ToString(), layerId);
  string response = client.UploadString(url, lineBuffer);
  XmlDocument doc = new XmlDocument();
  doc.LoadXml(response);
  XmlNode node = doc["LayerApi"];
  string s = node.InnerText;

  //
  // Handle an error situation
  //
  if (s.Contains("Error"))
  {
      throw new Exception(s);
  }

  //
  // Send an UPDATE command with a flyTo parameter
  //
  string lat, lon, zoom, rotation, angle;
  WebClient client = new WebClient();
  string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=update&id={1}&flyTo={2
},{3},{4},{5},{6}&instant=True", getIP().ToString(), layerId, lat, lon, zoom, rotation
, angle);
  string response = client.UploadString(url, lineBuffer);
  XmlDocument doc = new XmlDocument();
  doc.LoadXml(response);
  XmlNode node = doc["LayerApi"];
  string s = node.InnerText;

  //
  // Handle an error situation
  //
  if (s.Contains("Error"))
  {
      throw new Exception(s);
  }
```

For an example of the use of this command in the sample application, see flushBufferToWWT.

version

The **version** command returns the version number of the running version of the LCAPI.

Remarks

The version number should be used in an application to ensure that the LCAPI features used are supported by the version of the LCAPI running on the client computer.

Return Value

The following string will be included in the response if the call is successful (the version number will be more recent than the one shown below) :

```
<LayerApi><Version>2.8.21.1</Version></LayerApi>
```

There are no specific errors returned if the call is not successful.

Example Code

Refer to the getIP utility to see how to extract and check a version number.

General Parameters

The general parameters can be used with any of the commands.

Remarks

Parameter	Description
&datetime	Sets the viewing clock to the given date and time, in UTC format, for example: `html"1/1/2000 12:02:46 AM"`. This is the date and time that can be set through the **View** menu. Formats (month/day/year): "1/1/2010 11:00:00 PM" "1/1/2010 11:30 AM" "1/1/2010 11 am" "1/1/2000" "1/2000"
&timerate	The accelerated time to render the visualization, as a multiple of 10.
&flyto	Sets the position of the view camera. Requires five floating point numbers separated by commas: latitude, longitude, zoom level, rotation and angle. • Latitude is in decimal degrees, positive to the North. • Longitude is in decimal degrees, positive to the East. • Zoom level varies from 360 (the most distant view) to 0.00023 (the closest view). • Rotation is in radians, positive moves the camera to the left. • Angle is in radians, positive moves the camera forward. Optionally there can be a sixth parameter containing the frame to change the view to, which can be one of: **Earth, Moon, Mercury, Venus, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, Io, Ganymede, Callisto, Europa, Sun, ISS.**
&instant	Used with the **&flyto** parameter, set this to **true** to specify that the camera should jump to the location, or **false** that the camera should smoothly pan and zoom to the location.
&autoloop	True sets the layer manager to auto loop.

Example Code

Fly to a location

```
//
// Extract the values from the text boxes of a dialog, assuming there are rich text bo
xes with the names shown below.
//
string flyString = richLat.Text + "," + richLong.Text + "," + richZoom.Text + "," + ri
chRotation.Text + "," + richAngle.Text;

WebClient client = new WebClient();
string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=state&flyto={1}&instant=
{2}", getIP().ToString(), flyString, richInstant.Text);
string response = client.UploadString(url, "");
```

Utilities

The following utilities are not part of the API, but should be useful in building an application.

getIP

If the LCAPI application and WorldWide Telescope are running on the same computer, you can use the following utility to extract the IP address.

The **getIP** function extracts each IP address in turn, then calls the **CheckForWWTWebServer** function to check that WorldWide Telescope is both running at the given address and is recent enough that the LCAPI is supported.

Get IP address

```
        private IPAddress getIP()
        {
            IPAddress ipAddress = IPAddress.Loopback;

            // Find IPV4 Address

            foreach (IPAddress ipAdd in Dns.GetHostEntry(Dns.GetHostName()).AddressLis
t)
            {
                if (ipAdd.AddressFamily == AddressFamily.InterNetwork)
                {
                    if (CheckForWWTWebServer(ipAdd))
                    {
                        ipAddress = ipAdd;
                        break;
                    }
                }
            }
            return ipAddress;
        }

        private bool CheckForWWTWebServer(IPAddress address)
        {
            WebClient client = new WebClient();
            try
            {
                string version = client.DownloadString(string.Format("http://{0}:5050/
layerapi.aspx?cmd=version", address.ToString()));
                XmlDocument doc = new XmlDocument();
                doc.LoadXml(version);
                XmlNode node = doc["LayerApi"];

                if (node.OuterXml.Contains("Version"))
                {
                    // As version numbers are in the format xx.xx.xx.xx, an individual
 comparison of each
                    // number is necessary - a simple string comparison will give unre
liable results

                    string[] versionNumbers = node.InnerText.Split('.');
                    if (versionNumbers[0].CompareTo("2") >= 0 &&
                        versionNumbers[1].CompareTo("8") >= 0)
                        return true;
                }
            }
            catch (Exception ex)
            {
                // handle exception
            }
            return false;
        }
```

getUTCtime

With time series data care should be taken with the timings given in the data file. To convert local times to the recommended UTC time the following function could be used. Note that the local time is taken from the computer System time in the example below, so you may need to add some functionality to support local times that are not the same as your computer System time.

```
//
// Convert a string time to the correct format
//
private string getUTCtime(string time)
{
    DateTime d = DateTime.Parse(time);
    string s = d.ToUniversalTime().ToString();
    return s;
}
```

Sample Application

The following sample application loads earthquake data from a CSV file, sends it in buffers of up to 100 events, and displays it as a time series in WorldWide Telescope.

Sample program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

//
// Added References
//
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Xml;

//
//  Quakes is a sample of the WorldWide Telescope LCAPI
//

namespace Quakes
{
    public partial class Quakes : Form
    {
        public Quakes()
        {
            InitializeComponent();
            //
            // Set defaults
            //
            listBox1.SelectedItem = "1000000";
            listBox2.SelectedItem = "Earth";
        }

        private void buttonExit_Click(object sender, EventArgs e)
        {
            Dispose(true);
        }
```

```
        private void buttonBrowse_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                richFilename.Text = openFileDialog1.FileName;
            }
        }

        //
        // Add messages to the list box
        //
        private void addMessage(string m)
        {
            int i = richMessages.Items.Add(m);
            richMessages.SelectedIndex = i;
        }

        //
        // Go!
        //
        private void buttonGo_Click(object sender, EventArgs e)
        {
            string line;
            string w;

            richMessages.Items.Clear();

            try
            {
                initWWTLayer();

                StreamReader sr = new StreamReader(richFilename.Text);

                //
                // Ignore first line of data (the headings)
                //
                w = sr.ReadLine();

                //
                // Read lines of data until there are none left
                //
                while ((w = sr.ReadLine()) != null)
                {
                    line = parseLine(w,false);
                    flushBufferToWWT(line);
                }

                sr.Close();

                //
                // Flush buffer for last time - empty string indicates this is the las
t flush
```

```
                //
                flushBufferToWWT("");

                //
                // Record the end
                //
                addMessage("Data transmission ended: " + totalEvents + " events");
            }
            catch (Exception ex)
            {
                addMessage(ex.Message);
            }

        }

        //
        // Change the event color
        //
        private void buttonColor_Click(object sender, EventArgs e)
        {
            if (colorDialog.ShowDialog() == DialogResult.OK)
            {
                buttonColor.BackColor = colorDialog.Color;
            }
        }

        int timeSlot;                   // Data column number for time
        int latSlot;                    // Data column number for latitude
        int lonSlot;                    // Data column number for longitude
        int depSlot;                    // Data column number for depth
        int magSlot;                    // Data column number for magnitude
        string layerId;                 // string GUID for the WWT Layer
        string lineBuffer;              // Buffer to hold events until they are sent
        int lineCountInBuffer;          // Number of events in the buffer
        int totalEvents;                // Count of total events transmitted
        DateTime lastTimeBufferFlushed; // Time of last transmission, used to check fo
r time-out

        //
        // Locate the columns of the required data, they must be present but can be in
 any
        // order in the data file
        //

        private string initCSVData(string filename)
        {
            string w;           // line of data from data file

            timeSlot = -1;
            latSlot = -1;
            lonSlot = -1;
            depSlot = -1;
            magSlot = -1;
```

```
            //
            // First line (line 0) of the data file must contain the column headings
            //
            StreamReader sr = new StreamReader(filename);
            w = sr.ReadLine();
            string[] words = w.Split(new char[] { ',' });

            for (int i = 0; i < words.Length; i++)
            {
                if (words[i].IndexOf("Time", StringComparison.OrdinalIgnoreCase) != -1
)
                    timeSlot = i;
                else
                    if (words[i].IndexOf("Lat", StringComparison.OrdinalIgnoreCase) !=
-1)
                        latSlot = i;
                    else
                        if (words[i].IndexOf("Lon", StringComparison.OrdinalIgnoreCase
) != -1)
                            lonSlot = i;
                        else
                            if (words[i].IndexOf("Depth", StringComparison.OrdinalIgno
reCase) != -1)
                                depSlot = i;
                            else
                                if (words[i].IndexOf("Mag", StringComparison.OrdinalIg
noreCase) != -1)
                                    magSlot = i;
            }

            //
            // One of more heading is missing....
            //
            if (timeSlot == -1 || latSlot == -1 || lonSlot == -1 || depSlot == -1 || m
agSlot == -1)
                throw new Exception("Source file does not contain one or more of: time
, lat, lon, depth, mag");

            //
            // Record a successful reading of the headings
            //
            addMessage("Time = " + timeSlot.ToString() + " Lat = " + latSlot.ToString(
) + " Lon = " + lonSlot.ToString() + " Depth = " + depSlot.ToString() + " Mag = " + ma
gSlot.ToString());

            //
            // Extract the start time from the data file by reading line 1
            //

            w = sr.ReadLine();
            string startDate = parseLine(w, true);
            sr.Close();
```

```
            //
            // Record a successful reading of the start
            //
            addMessage("Start time = " + startDate);

            return startDate;
        }

        //
        // Convert a string time to the correct format
        //
        private string getUTCtime(string time)
        {
            DateTime d = DateTime.Parse(time);
            string s = d.ToUniversalTime().ToString();
            return s;
        }

        //
        // Parses a line from the data file
        //
        private string parseLine(string w, bool timeOnly)
        {
            string[] words = w.Split(new char[] { ',' });
            string line = "";

            string time = getUTCtime(words[timeSlot]);
            string lat = words[latSlot];
            string lon = words[lonSlot];
            string depth = words[depSlot];
            string mag = words[magSlot];

            if (timeOnly)
            {
                return time;
            }
            line = time + "\t" + lat + "\t" + lon + "\t" + depth + "\t" + mag + "\t";

            return line;
        }

        //
        // Create a new WWT layer
        //
        private void <a name="initWWTLayer">initWWTLayer()
        {
            //
            // Send a NEW command, extracting info from the data file and the UI
            //
            WebClient client = new WebClient();
            string startDate = initCSVData(richFilename.Text);
            string name = Path.GetFileNameWithoutExtension(richFilename.Text);
```

```
            string rate = listBox1.SelectedItem.ToString();
            string frame = listBox2.SelectedItem.ToString();
            string color = buttonColor.BackColor.ToArgb().ToString("X8"<font size="2">
);
            string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=new&datetime
={1}&timerate={2}&name={3}&frame={4}&color={5}",  getIP().ToString(), startDate, rate,
 name, frame, color);
            // field string below is delimited by tabs, not spaces
            string response = client.UploadString(url, "TIME    LAT    LON    DEPTH
 MAG");
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(response);
            XmlNode node = doc["LayerApi"];
            XmlNode child = node.ChildNodes[0];
            layerId = child.InnerText;

            //
            // Handle an error situation
            //
            if (layerId.Length != 36)
                throw new Exception("Invalid Layer Id received");

            //
            // Record a successful creation of a new layer
            //
            addMessage("New layer Id = " + layerId);

            //
            // Clear the buffer
            //
            lineBuffer = string.Empty;
            lineCountInBuffer = 0;
            totalEvents = 0;
            lastTimeBufferFlushed = DateTime.Now;
        }

        //
        // Utility to extract IP address
        //
        private IPAddress getIP()
        {
            IPAddress ipAddress = IPAddress.Loopback;

            // Find IPV4 Address

            foreach (IPAddress ipAdd in Dns.GetHostEntry(Dns.GetHostName()).AddressLis
t)
            {
                if (ipAdd.AddressFamily == AddressFamily.InterNetwork)
                {
                    if (CheckForWWTWebServer(ipAdd))
                    {
                        ipAddress = ipAdd;
```

```
                        break;
                    }
                }
            }
            return ipAddress;
        }

        private bool CheckForWWTWebServer(IPAddress address)
        {
            WebClient client = new WebClient();
            try
            {
                string version = client.DownloadString(string.Format("http://{0}:5050/
layerapi.aspx?cmd=version", address.ToString()));
                XmlDocument doc = new XmlDocument();
                doc.LoadXml(version);
                XmlNode node = doc["LayerApi"];

                if (node.OuterXml.Contains("Version"))
                {
                    // As version numbers are in the format xx.xx.xx.xx, an individual
 comparison of each
                    // number is necessary - a simple string comparison will give unre
liable results

                    string[] versionNumbers = node.InnerText.Split('.');
                    if (versionNumbers[0].CompareTo("2") >= 0 &&
                        versionNumbers[1].CompareTo("8") >= 0)
                        return true;
                }
            }
            catch (Exception ex)
            {
                // handle exception
            }
            return false;
        }


        //
        // Buffer the data and send it to WWT every N events or every T time units
        //
        private void <a name="flushBufferToWWT">flushBufferToWWT(string line)
        {
            const int FlushThresholdInEventCount = 100;     // Flush the buffer when t
his number of events is stored
            const int FlushThresholdInSeconds = 2;          // Flush the buffer after
this number of seconds has elapsed
            //
            // If line length is zero, this is the last transmission
            //
            if (line.Length > 0)
            {
```

```
                lineBuffer = lineBuffer + line + Environment.NewLine;
                lineCountInBuffer++;
            }
            DateTime now = DateTime.Now;

            //
            // Send the buffer if flushing for the last time OR the buffer is full OR
the timeout has been reached
            //
            if ((line.Length == 0  lineCountInBuffer > 0) ||
                lineCountInBuffer == FlushThresholdInEventCount ||
                now.Subtract(lastTimeBufferFlushed) > TimeSpan.FromSeconds(FlushThresh
oldInSeconds))
            {
                //
                // Send an UPDATE command
                //
                WebClient client = new WebClient();

                string url = string.Format("http://{0}:5050/layerApi.aspx?cmd=update&i
d={1}", getIP().ToString(), layerId);
                string response = client.UploadString(url, lineBuffer);
                XmlDocument doc = new XmlDocument();
                doc.LoadXml(response);
                XmlNode node = doc["LayerApi"];
                string s = node.InnerText;

                //
                // Handle an error situation
                //
                if (s.Contains("Error"))
                {
                    throw new Exception(s);
                }
                //
                // Record a successful transmission
                //
                addMessage(now.ToString() + ": " + lineCountInBuffer.ToString() + " ev
ents sent");
                totalEvents += lineCountInBuffer;

                //
                // Clear the buffer
                //
                lineBuffer = string.Empty;
                lastTimeBufferFlushed = now;
                lineCountInBuffer = 0;
            }
        }
    }
}
//
// End
```

```
//
```