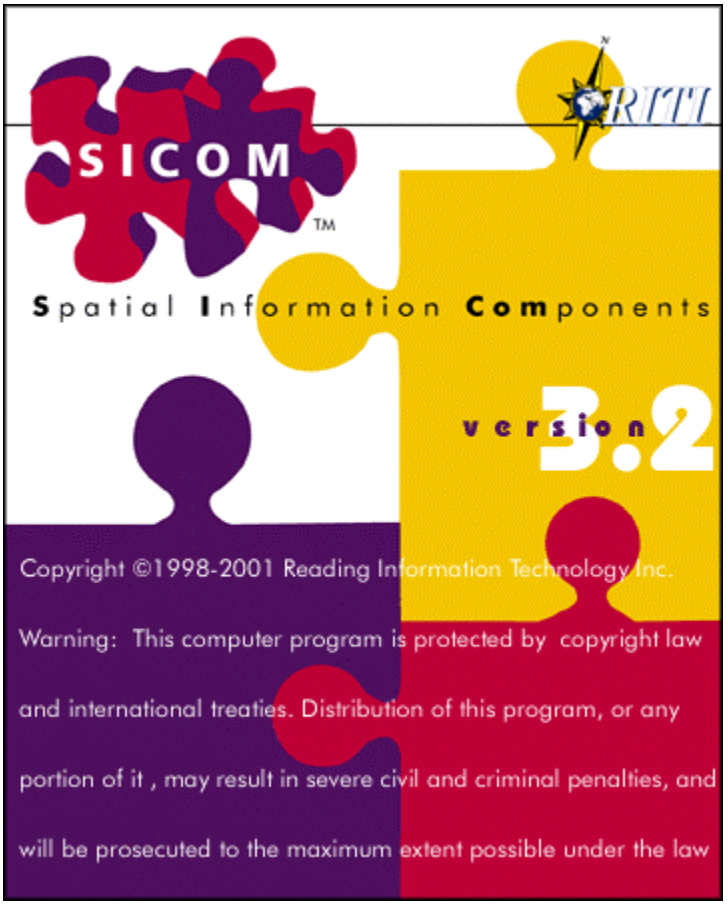
SPATIAL INFORMATION COMPONENTS

Working with SICOM3.2

Visual Basic Tips



January 5, 2002
274 Main Street, Suite 302
Reading, MA 01867

Tel./ 781.942.1655
Fax/ 781.942.2161
Website/ www.riti.com
E-mail/ riti@riti.com

READING INFORMATION TECHNOLOGY INC.

Table of Content

I. Introduction

This document presents information required to effectively use the SICOM DisplayModule (SDM) to provide GIS services to a Visual Basic application. The material comprises five sections, as follows:

- Creating a Standard Client
- Creating an ActiveX Client
- Avoiding Client/Server Communications Lockup
- Handling Multiple Clients in One Application
- Examples

In the material that follows, where a phrase or a word has been put in italic, a generic name has been used. A name of your choosing should be substituted. The code snippets in this document can also be copied and pasted in to a Visual Basic project.

II. Creating a Standard Client

In this Section, the essentials of creating a simple client are covered, along with tips on how to make the overall application work smoothly as a unit. These topics are

- Connecting to the SDM
- Communicating with the SDM
- Floating Forms
- Making Message Boxes Modal

Connecting to the SDM:

a. Choose which form will be the main/startup form.

b. If no environment variables are defined for the required support files (See SICOM Developer's Guide – Runtime Environment), the SDM will need to set internal environment variables to point to files in a Support directory. When the client is launching the SDM (as in this case), the client must supply code in the form_load of the startup form that tells the SDM where to find the support directory.

Ex.

```
'Writes to a text file that tells the SDM where to look for file support
Open "C:\temp\RMSdataDir.txt" For Output As 10
Path = ".,e:\sicom3.2\sdmtest"
Print #10, Path
Close 10
```

Note: This code snippet assumes that the system drive is C, and that there is a support directory parallel to the directory given by Path.

IMPORTANT: examples 1, 2, 4, 5, & 6 illustrated this concept. Before running one of these, open the project and change the path used in the "Open" statement and the variable "Path" used in the "Print" statement to something meaningful for the host machine. In particular, set "Path" to point to the VC6 directory, where ever it is installed.

c. Go to General Declarations and declare a variable to hold the SDM reference.

Ex.

```
'Declare variable to hold the SDM reference
Public pgmAPI as Object
```

Note: Make the variable public rather than declaring it with a Dim statement so that it can be referenced in other forms if necessary.

d. In form_load of the main form, establish the connection to the server.

Ex.

```
'Brings up SDM
Set pgmAPI = CreateObject("SDM.ProgramAPI")
```

Note: The string in parentheses is the registered name of the Display Module top-level API.

Communicating with the SDM

Once the *pgmAPI* object has been set, it can be used to send any messages to the SDM that are documented in the section entitled "ProgramAPI". If a method takes no parameters and returns nothing, the message is sent as follows:

pgmAPI.methodName

where *methodName* is the name of one of the methods documented.

e.g. *pgmAPI*.orderFront

If a method takes arguments, they are simply listed after the method name, as follows:

pgmAPI.methodName arg1, *arg2*, …

e.g. *pgmAPI*.moveMainWnd 23, 34, 50, 12

Of course, the arguments must match the documented types.

If the method returns a value, then the arguments must be in parentheses and separated by comma, rather than as described above. The return value may be assigned to a variable:

e.g. *result* = *pgmAPI*.convertFromDecimalToGeoString(85.5, true)

Other API objects are accessible via the ProgramAPI, and through those still others (See SICOM Developer's Guide – SDM Services).

Floating Forms

When using the SDM, it is usually desirable to show the map view as large as possible. A good strategy, therefore, is to have client-generated forms appear as auxiliary dialogues of the SDM, i.e. have them be relatively small and float above the SDM window while they are active. In this way, the user can click on the map without causing the SDM to cover up the dialogue. To achieve this look, the following must be placed in each form that is to be floated.

a. Add the following to the General Declarations of the form

```
'Will keep the client dialogs on top of the display module
Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, _
ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, _
ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Const HWND_TOPMOST = -1
Const SWP_SHOWWINDOW = &H40
```

b. In each form_load subroutine enter the following code:

```
'To keep the client forms on top of the display module
Dim retValue As Long

retValue = SetWindowPos(Me.hwnd, HWND_TOPMOST, _
        Me.CurrentX, Me.CurrentY, 200, 150, SWP_SHOWWINDOW)
```

Note: Me.CurrentX and Me.CurrentY, controls the position of the form on the monitor. They can be replaced with pixel values chosen by the programmer. The values of 200 and 150 are the default values determining the size of the form. These values can also be replaced, but first the ScaleMode property of the form must be changed from **1-Twip** to **3-Pixel**. The 200 can then be replaced with ***form1.S*caleWidth** and the 150 replaced with ***form1.*ScaleHeight**. Small adjustments might have to be made to these.

Making Message Boxes Modal:

After the prompt has been typed in, the buttons argument becomes active. This is where the message box can be made to float or in this case become modal. There are two ways to make the message box modal.

1) **VbApplicationModal**. This option means, "Application modal; the user must respond to the message box before switching to any of the Forms of the current application". The numeric value of this option is 0.
2) **VbSystemModal**. The description of this option is, "System modal; all applications are suspended until the user responds to the message box". The numeric value of this option is 4096.

Note: Each of these options can be chosen by their numeric values. Making the message box modal will also force it to float on top of all forms and the display module.

III. Creating an ActiveX Client

The SDM has the capability to send messages back to a client, i.e. two-way communication (asynchronous). Taking advantage of this capability can produce more flexible applications with less effort. To utilize this capability, the client must expose API methods expected by the SDM (See SICOM Developer's Guide - Client Services). In the Visual Basic framework this means that the ActiveX project format must be chosen from the startup dialog. (If two-way communication is not needed, open the Standard project format option from the dialog box.) Note that all discussions under "Creating a Standard Client" are valid for an ActiveX client. This section discusses additional requirements and tips, as follows:

- Adding a Form
- Providing the SDM with access to the Class Module
- Creating the Main Subroutine
- Setting Project Properties
- Exposing Methods

Adding a Form

When starting up an ActiveX project, VB automatically creates a class module (in which to expose methods), but no forms. A form must be added to the project in order to establish a connection with the display module, even if no client forms will be seen when the project first starts up (The way to hide the form will be discussed next).

To create the form, go to Project → Add Form.

Providing the SDM with Access to the Class Module:

To provide the SDM with the name of the module to which it will send return messages, add:

mmAPI.setReceiver "*ProjectName.ClassModuleName*"

Note: This line of code must occur after the MouseModeAPI reference has been initialized, via getMouseModeAPI. The string "*ProjectName.ClassModuleName*" is registry name of the class module. (See Setting Project Properties, Project Description).

Creating the Main Subroutine

This subroutine is needed in order to decide what will be the startup form. To create subroutine Main…

a. Go to Project → Add Module. The new module will appear on the screen.

b. Go to Tools→ Add Procedure.

c. In the text field type Main click OK. The module will now be open to the newly created procedure. Which will look like the following:

```
Public Sub Main ()
End Sub
```

d. Delete Public from the top line, so that only Sub Main () appears.

e. Add the following code to determine the startup form:

```
If App.StartMode = vbModeStandAlone then
        Form1.show
End if
```

Note: If no forms are to be shown at startup then replace "show" with "hide". The startup form must be the one in which the SDM server connection is made.

Setting Project Properties:

a. Go to: Project → Properties

b. Enter the name of the project and choose Main as the startup object from the drop-down list.

c. Under the heading, Project Description, type the name of the project, space, then the name given to the class module. (This is the string that is needed when establishing the connection between the Display Module and the Class Module).

d. Choose the Component tab and select the StandAlone option under the StartMode heading.

e. Choose the Debugging tab. Deselect "Use existing browser".

f. Click OK.

Exposing Methods

The class module was created automatically by VB due to the selection of ActiveX as the project type. This class module is the object that must expose the methods prescribed by the SDM. (See section "Client Services" in the documentation.) To expose a method…

a. Select the class module then got to: Tools → Add Procedure.

b. In the text field, type in the exact name of the method.

Note: If there are any arguments that go with this method, make sure they are declared within the parentheses of the method.

c. Make sure "public" is selected under the Scope heading.

d. Press OK. The method has now been exposed and is accessible through the class module.

e. Add your application-specific code to the method.

IMPORTANT: in the properties for the class module, set the "Instancing" property to "5 – MuliUse" or "6 – GlobalMultiUse".

IV. Handling Multiple Clients in One Application

VB Client processes provide a way to extend and customize the SDM. Several clients, each providing one extension or customization, can be seamlessly incorporated into a single application. The simplest way to accomplish this is to create each client as a tool or manager (See SICOM Developer's Guide – SDM Tools and Managers). This is accomplished as follows:

a. In the class module, expose the methods "initialize", "viewOpened", "viewClosed", and "restoreWnd".

   IMPORTANT: add the "initialize" method as a function, rather than a subroutine. To do this, select the Function option button in the dialog box.

b. Add the CreateObject for the ProgramAPI to "initialize" (instead of the form_load subroutine in the main form). Also, add any other calls used to set up access to/from the SDM, i.e. *mmAPI*.setReceiver.

   IMPORTANT: before the end of the function

   1. Show the initial form via *formName*.show
   2. Set the function's return value to true via initialize = true

c. Add code to the "viewClosed" method to make the appropriate adjustments, if and when the user closes a view. The SDM automatically calls this method whenever a view closes. In the demos of examples 3 and 7, the managers associate themselves with the active view, only (in the "initialize" method). If the user closes this view, the manager shuts itself down. Before doing so, the manager must notify the SDM that it is about to shut down. Thus the code in the "viewClosed" method is as follows:

   Unload *formName*

frmMain.pgmAPI.notifyShutDown "*ProjectName.ClassModuleName*"

where the string *ProjectName.ClassModuleName* is the same as that used in MouseModeAPI.setReceiver.

d. Add code to the "viewOpend" method to make the appropriate adjustments, if and when the user opens a view. The SDM automatically calls this method whenever a view is opened. In the demos of examples 3 and 7, the managers operate only on the view that was active at manager start-up. Thus, they are not concerned with other views the user may open. Therefore, the "viewOpened" method contains no code. However, the method must still be exposed or the SDM will produce an error if the user opens a view.

e. Add code to the termination routine, i.e. the routine associated with the manager's Exit button, to notify the SDM that the manager is about to shut down. This is handled the same way as in step c, above. However, there is one issue to resolve. It is necessary to run the manager once in stand-alone mode, to register it. If the notifyShutDown statement in the termination routine is unconditional, the manager will fail when the user clicks the Exit button. Although this is inconsequential, since the end result is the termination of the manager, it is ugly. To avoid this, condition the notifyShutDown call on whether the ProgramAPI reference has been initialized, as follows:

```
If Not pgmAPI Is Nothing Then
  pgmAPI.notifyShutDown " ProjectName.ClassModuleName "
End If
```

f. Run each client once to get them registered.

g. Create the "mngrList.text" file. The first line in the file is the title of the menu and the following lines provide the names of the menu items. Each of these lines contains the registry name of the manager and the corresponding menu item name. These are separated by a comma. Below is the "mngrList.text" file for example 7.

Ex.

```
Examples
Example2a.RcvServer, SelectFeature
Example4a.RcvServer, Add/Delete
Example6a.RcvServer, GPS
```

Note: The file "mngrList.text" must be located in the support directory if there is no global SDM_MNGR_LIST environment variable.

The SDM will launch the corresponding client in response to a selection on the menu created above.

IMPORTANT: examples 3 illustrates these principles in one client, while example 7 shows a more realistic case by modifying examples 2, 4, and 6 to become managers.

Before running one of these, copy the "mngrList.text" file from the corresponding project directory to the "Support" directory. Be sure to remove the "mngrList.text" file from the "Support" directory when through with the demo (so that these demo managers will no longer appear on the SDM menu.)

V. Examples

All of the principles discussed above are demonstrated in a series of examples that also demonstrate how to effectively use particular groups of SDM API functions to accomplish particular objectives. These examples build in complexity, as described below.

Example 1

This is the simplest example, demonstrates the fundamentals discussed in Section II, above, while also demonstrating three techniques for synchronous (one-way) feature selection. Each of the three techniques makes use of one of the following methods:

- MouseModeAPI.setSelectPointFeatureMode
- MouseModeAPI .setSelectSingleFeatureMode
- MouseModeAPI .setSelectFeatureCollectionMode

supported by the following methods:

- MouseModeAPI .isSynchronousOpComplete
- MouseModeAPI .getRespondingViewIndex
- MouseModeAPI .getX
- MouseModeAPI .getY
- MapAPI.getSelectedFeatures

The example also demonstrates the ProgramAPI.exitApp message.

IMPORTANT: before running this example, modify the paths used in the creation of the "RMSdataDir.txt" file, as described in Section II.

Example 2

This demonstrates the principles of ActiveX projects discussed in Section III, while also demonstrating three techniques for asynchronous (two-way) feature selection. This example looks to the end-user very much like Example 1, but it demonstrates how simple it is to remain in a particular selection mode continuously, via two-way communication. The client ends a mode when the user clicks the "Clear" button. Each of the three techniques makes use of one of the following methods:

- MouseModeAPI .setSelectPointFeatureMode
- MouseModeAPI .setSelectSingleFeatureMode
- MouseModeAPI .setSelectFeatureCollectionMode

supported by the following methods:

- MapAPI.getSelectedFeatures
- MapAPI.querySelectionByIndex

The example also uses the client methods:

- receiveLocation – in response to setSelectPointFeatureMode
- receiveSelectionViewIndex – in response to setSelectSingleFeatureMode and setSelectFeatureCollectionMode

IMPORTANT: before running this example, modify the paths used in the creation of the "RMSdataDir.txt" file, as described in Section II.

Example 3

This is essentially Example 2, along with the principles discussed in Section V, i.e. accessing a VB client via the SDM menu. Thus, this example demonstrates, for the first time, the client methods:

- viewOpened
- viewClosed
- restoreWnd
- initialize

IMPORTANT: before running this example, copy the file "mngrList.text" from the project directory "Ex3" to the "Support" directory. After finishing with the demo remove this file from the support directory to prevent the menu associated with this demo from appearing.

Example 4

This is a client that incorporates adding/deleting point, text, and line features. The point and text features are handled synchronously, while the line features are handled asynchronously. Thus, the example shows how synchronous and asynchronous techniques can be employed within the same client. This example also allows the user to switch from geodetic coordinates to projected coordinates, thereby demonstrating how to override the default Exchange Coordinate System (See SICOM Developer's Guide). This example uses, for the first time, the client method:

- receiveLocList

IMPORTANT: before running this example, modify the paths used in the creation of the "RMSdataDir.txt" file, as described in Section II.

Example 5

This is a synchronous client that shows how to connect to a GPS and use the information from the GPS to move an icon on a map. This example also shows how the client can load a map into the SDM, via ViewManagerAPI.openFromFile.

IMPORTANT: before running this example, modify the paths used in the creation of the "RMSdataDir.txt" file, as described in Section II, and change the path used as argument to ViewManagerAPI.openFromFile to correctly reference the file on the host platform. **This demo will not run without the SICOM GPSreader module, which is not included with the developer installation**. Contact RITI for information about GPSreader.

Example 6

This is an asynchronous version of Example 5 and it utilizes the procedure of raising an event. The latter avoids a potential lock-up condition between the client and the GPSreader.

IMPORTANT: before running this example, modify the paths used in the creation of the "RMSdataDir.txt" file, as described in Section II, and change the path used as argument to ViewManagerAPI.openFromFile to correctly reference the file on the host platform. **This demo will not run without the SICOM GPSreader module, which is not included with the developer installation**. Contact RITI for information about GPSreader.

Example 7

This involves examples 2, 4, and 6, which are seamlessly incorporated into the SDM via adding a menu to the SDM. Examples 2, 4, and 6 have been slightly modified to work as managers, and thus resemble the structure of Example 3. Examples 2, 4, and 6 use the four client methods listed under Example 3 and the principles discussed under section V.

IMPORTANT: before running this example, copy the file "mngrList.text" from the project directory "Ex7" to the "Support" directory. This will provide a new menu with three items, one for each of the examples 2, 4, and 6. **The third menu item (which launches example 6) will not run without the SICOM GPSreader module, which is not included with the developer installation**. (Contact RITI for information about GPSreader.) After finishing with the demo remove this file from the support directory to prevent the menu associated with this demo from appearing.