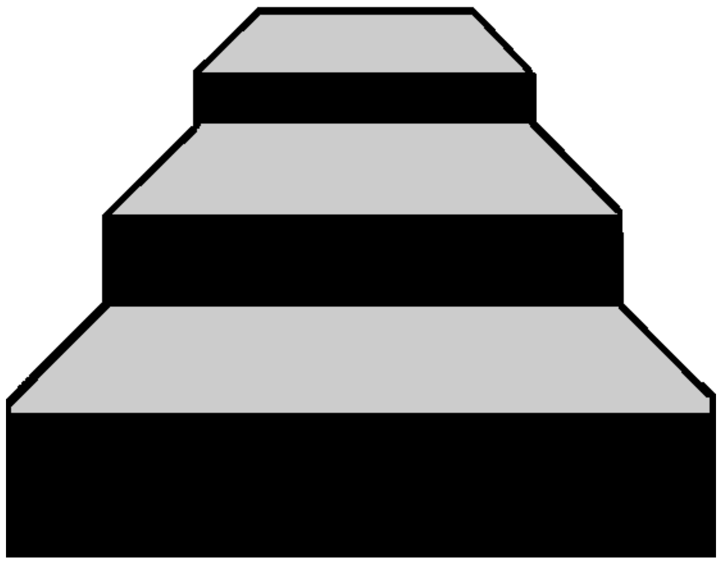
ST-Viewer™ OLE Automation Reference Manual



STEP Tools® Software

Contributors

Written by Alexey Lipatov.

STEP Tools, Inc.
14 First Street
Troy, New York 12180

Phone	**(518) 687-2848**
Fax	**(518) 687-4420**
E-Mail	**info@steptools.com**
Web.	**http://www.steptools.com**

Contents

Contents . . . . **iii**

1 — Introduction to OLE Automation **1**
1.1 Introduction 1
1.2 What is Automation? 1

2 — The ISTview interface **3**
2.1 Methods 3

3 — The ISTviewEvents Interface **9**
3.1 Methods 9

4 — Examples **11**
4.1 Overview 11
4.2 The Visual C++ example 11
4.2.1 How the driver works 13
4.3 The Visual Basic example 16
4.3.1 How the driver works 16
4.4 The Microsoft Excel (VBA) example 17
4.4.1 How the Macros Work 18

1 Introduction to OLE Automation

1.1 Introduction

The ST-Viewer automation interface is designed so users can easily add STEP viewing capability to their applications. The interface allows an external application to use ST-Viewer as the viewing platform, and to communicate with ST-Viewer via a series of events generated from ST-Viewer.

1.2 What is Automation?

Automation is a mechanism to manipulate an application's objects from outside the application. To make this happen, the application exposes one or more COM interfaces. These interfaces can contain methods and properties. These become accessible to the automation client, and the exposing application becomes the automation server.

The interfaces are described in ODL (Object Description Language) and presented to the client as a type library file. Development tools or wizards can then import the type library and generate a code framework for the automation client. The automation client can be written in many languages, including Microsoft Visual Basic, Visual C++, and Visual J++.

ST-Viewer exposes the **ISTview** interface to allow an external application to control it. It also uses another interface **ISTviewEvents** to send events to an external application. The latter interface is not exposed by the viewer, but should be exposed by an external application. In this case the viewer acts as an automation client, and the external application acts as an automation server.

The ST-Developer distribution includes an example driver program that controls ST-Viewer and receives events. The examples are written in Microsoft Visual C++, Visual Basic and Visual Basic for Applications (using Microsoft Excel) .

This reference manual contains descriptions of the ST-Viewer COM interfaces and examples of their use. It is assumed that the reader is familiar with COM and has C++ and/or Visual Basic programming experience.

2 The ISTview interface

2.1 Methods

The ISTview interface is the interface that ST-Viewer exposes to allow user programs to control it. This interface has only methods and no properties. The following is the list of the ODL definitions for these methods.

```
BOOL LoadFile(
             BSTR strName
             );
```

Use this method to load a STEP Part 21 file with the specified name. This method returns **TRUE** if the file was successfully loaded, or **FALSE** if it was unable to load the file.

```
void SetWindowPosition(
             short nX,
             short nY,
             short nWidth,
             short nHeight,
             long nShow,
             short nSeparatorOffset
             );
```

This method changes the position and size of the STEP model window. The first four arguments set the position and size of the window. The **nShow** specifies the window sizing and positioning flags. The low-order word is the same as the **uFlags** argument to the **SetWindowPos** Win32 API function and can be a boolean **OR** combination of the following values.

`SWP_DRAWFRAME`
Draws a frame (defined in the window's class description) around the window.

`SWP_FRAMECHANGED`
Sends a **WM_NCCALCSIZE** message to the window, even if the window's size is not being changed. If this flag is not specified, **WM_NCCALCSIZE** is sent only when the window's size is being changed.

`SWP_HIDEWINDOW`
Hides the window.

`SWP_NOACTIVATE`
Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost group.

`SWP_NOCOPYBITS`
Discards the entire contents of the client area. If this flag is not specified, the valid contents of the client area are saved and copied back into the client area after the window is sized or repositioned.

`SWP_NOMOVE`
Retains the current position (ignores the **nX** and **nY** arguments).

`SWP_NOOWNERZORDER`
Does not change the owner window's position in the Z order.

`SWP_NOREDRAW`
Does not redraw changes. If this flag is set, no repainting of any kind occurs. This applies to the client area, the nonclient area (including the title bar and scroll bars), and any part of the parent window uncovered as a result of the window being moved. When this flag is set, the application must explicitly invalidate or redraw any parts of the window and parent window that need redrawing.

`SWP_NOREPOSITION`
Same as the **SWP_NOOWNERZORDER** flag.

`SWP_NOSENDCHANGING`
Prevents the window from receiving the **WM_WINDOWPOSCHANGING** message.

`SWP_NOSIZE`
Retains the current size (ignores the **nWidth** and **nHeight** arguments).

`SWP_NOZORDER`
Retains the current Z order.

`SWP_SHOWWINDOW`	Displays the window.

The high-order word is the same as the **nCmdShow** argument to the **ShowWindow** Win32 API function and can be one of the following values.

`SW_HIDE`	Hides the window and activates another window.
`SW_MAXIMIZE`	Maximizes the specified window.
`SW_MINIMIZE`	Minimizes the specified window and activates the next top-level window in the Z order.
`SW_RESTORE`	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
`SW_SHOW`	Activates the window and displays it in its current size and position.
`SW_SHOWDEFAULT`	Sets the show state based on the **SW_** flag specified in the **STARTUPINFO** structure passed to the **CreateProcess** function by the program that started ST-Viewer.
`SW_SHOWMAXIMIZED`	Activates the window and displays it as a maximized window.
`SW_SHOWMINIMIZED`	Activates the window and displays it as a minimized window.
`SW_SHOWMINNOACTIVE`	Displays the window as a minimized window. The active window remains active.
`SW_SHOWNA`	Displays the window in its current state. The active window remains active.
`SW_SHOWNOACTIVATE`	Displays a window in its most recent size and position. The active window remains active.

SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position.

The **nSeparatorOffset** sets the width of the identification pane within the document window.

If any of the dimensional arguments is -1, this window parameter remains unchanged. For example:

```
SetWindowPosition(
        0, 0, -1, 456,
        MAKELONG(SWP_SHOWWINDOW, SW_SHOWNORMAL), -1
        );
```

will move the model window to the upper left corner of the ST-Viewer MDI client window, set its height to 456 pixels and leave the width of the model window and the identification pane unchanged.

```
void SetMainWindowPosition(
            short nX,
            short nY,
            short nWidth,
            short nHeight,
            long nShow
            );
```

This method is similar to the previous one, but it changes the size and position of the ST-Viewer main window.

```
void RotateThumbWheel(
            short nWheel,
            short nDegrees
            );
```

The method "rotates" the specified thumbwheel by a number of degrees. The thumbwheels perform different functions depending on the current scene viewer. The number of thumbwheels also vary, but only first three are supported at this time. The **nWheel** argument specifies the thumbwheel.

0	Left
1	Bottom
2	Right
3	Top left (not supported)

```
void Zoom(
             float fZoom
             );
```

Use this method to zoom the camera in and out. The **fZoom** value specifies the same value as the zoom scrollbar in the geometric view.

```
void MakeSelected(
             long nID,
             long nInstance,
             short bSelect
             );
```

This method selects an object in the model identified by its STEP entity ID (the **nID** argument) and the instance number (**nInstance**). A model can have several instances of the same STEP object, for example an assembly can have several parts that share the same geometry. The instances are counted starting from 1. Set **bSelect** to **TRUE** to select the object, set it to **FALSE** to cancel the selection.

```
void MakeAllSelected(
             long nID,
             short bSelect
             );
```

Use this method to select all instances of the object with the specified STEP ID. A model can have several instances of the same STEP object, for example an assembly can have several parts that share the same geometry. The **nID** argument specifies the STEP entity ID of the object from the original file, and the **bSelect** argument should be set to **TRUE** to select all instances or to **FALSE** to cancel selection.

```
void MakeVisible(
             long nID,
             long nInstance,
             short bVisible
             );
```

Use this method to display or hide an instance of an object in the model. The object is identified by its STEP entity ID (the **nID** argument) and the instance number (**nInstance**). To display the object set **bVisible** to **TRUE**, to hide the object set it to **FALSE**.

```
void MakeAllVisible(
             long nID,
             short bVisible
             );
```

Use this method to display or hide all instances of the object with the specified STEP ID. The **nID** argument specifies the STEP entity ID of the object from the original file, and the **bVisible** argument should be set to **TRUE** to display all instances or to **FALSE** to hide them.

```
void SetColor(
            long nID,
            long nInstance,
            double dRed,
            double dGreen,
            double dBlue);
```

Use this method to set the color of the specified object. The object is identified by its STEP entity ID (the **nID** argument) and the instance number (**nInstance**). The last three arguments are red, green, and blue components of the color, expressed as the values within the 0...1 range. For example, a default gray color will be expressed as (0.8, 0.8, 0.8).

3 The ISTviewEvents Interface

3.1 Methods

The **ISTviewEvents** interface is the interface that the user program should implement to allow ST-Viewer send events to this program. ST-Viewer uses the connection points mechanism to connect to the implementation in the user program.

This interface has only methods and no properties. The following is the list of the ODL definitions for these methods.

```
void OnObjectDisplay(
             long nID,
             long nInstance,
             short bDisplay
             );
```

This method is called by ST-Viewer when an object is being displayed or hidden. The object is identified by its STEP entity ID (the **nID** argument) and the instance number (**nInstance**). A model can have several instances of the same STEP object, for example an assembly can have several parts that share the same geometry. The instances are counted starting from 1. The **bDisplay** argument is set to **TRUE** when the object is being displayed or **FALSE** when the object is being hidden.

```
void OnObjectSelect(
             long nID,
             long nInstance,
             short bSelect
             );
```

This method is called by the viewer when the selection state of an object changes.

The object is identified by its STEP entity ID (the **nID** argument) and the instance number (**nInstance**). A model can have several instances of the same STEP object, for example an assembly can have several parts that share the same geometry. The instances are counted starting from 1. The **bSelect** argument is set to **TRUE** when the object is being selected or **FALSE** when the selection is being canceled.

4 Examples

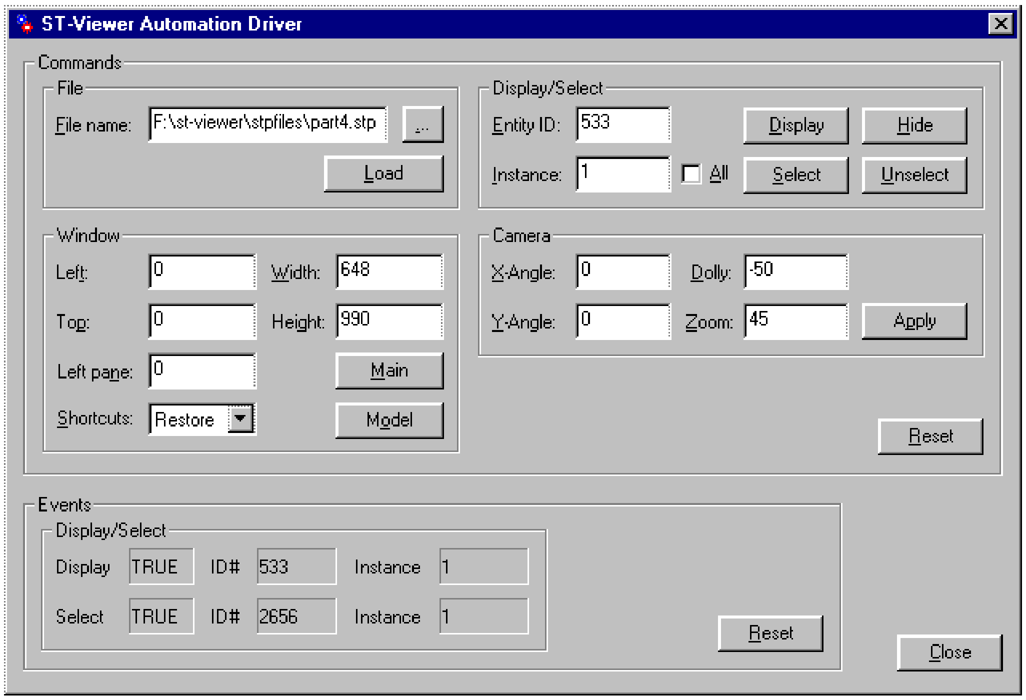
4.1 Overview

ST-Developer includes three examples of a driver program that controls ST-Viewer and receives its events. The examples are written in Microsoft Visual C++, Visual Basic and Visual Basic for Applications (VBA). The first two programs are distributed as Microsoft Visual Studio projects and include full source code. The VBA example is distributed as a Microsoft Excel template file. They are installed in the **automation** directory under the ST-Developer root installation directory.

4.2 The Visual C++ example

The window of the ST-Viewer Automation Driver is shown in Figure 4.1. The top portion of the window (the **Commands** area) contains controls that use the methods of the **ISTview** interface.

`File name`	Specify the name of the file you want to load into the viewer.
`[...]`	Display the **Open File** dialog box to locate the file you want to load into the viewer.
`Load`	Invoke the **LoadFile** method.
`Entity ID`	Specify the entity ID for the object you want to display, hide, or select/unselect.


ST-Viewer Automation Driver

Commands

File
File name: F:\st-viewer\stpfiles\part4.stp

Display/Select
Entity ID: 533
Instance: 1
☐ All

Window
Left: 0
Width: 648
Top: 0
Height: 990
Left pane: 0
Shortcuts: Restore

Camera
X-Angle: 0
Dolly: -50
Y-Angle: 0
Zoom: 45

Events

Display/Select
Display TRUE ID# 533 Instance 1
Select TRUE ID# 2656 Instance 1

Figure 4.1 — ST-Viewer Automation Driver window

`Instance`	Specify the instance number for the object specified in the Entity ID box.
`All`	Apply the display/select commands to all instances of the specified object.
`Display`	Display the object(s) with the specified entity ID and instance(s). (Invoke the **MakeVisible** or **MakeAllVisible** methods.)
`Hide`	Hide the specified object(s). (Invoke the **MakeVisible** or **MakeAllVisible** methods.)
`Select`	Select the specified object(s). (Invoke the **MakeSelected** or **MakeAllSelected** methods.)
`Unselect`	Cancel the selection of the specified object(s). (Invoke the **MakeSelected** or **MakeAllSelected** methods.)
`Left`	Specify the horizontal position of the window.
`Top`	Specify the vertical position of the window.
`Width`	Specify the width of the window.
`Height`	Specify the height of the window.

`Left pane`	Specify the width of the left (identification) pane in the model window.
`Shortcuts`	Select a window shortcut command. The available commands are: **None**, **Minimize**, **Maximize**, **Restore**. When **None** is selected, the window position and size values are used. The other commands ignore these values.
`Main`	Invoke the **SetMainWindowPosition** method.
`Model`	Invoke the **SetWindowPosition** method.
`X-Angle`	Specify the number of degrees to rotate the left thumbwheel.
`Y-Angle`	Specify the number of degrees to rotate the bottom thumbwheel.
`Dolly`	Specify the number of degrees to rotate the right thumbwheel.
`Zoom`	Specify the **fZoom** argument value in the **Zoom** method.
`Apply`	Invoke the **RotateThumbWheel** and **Zoom** methods.
`Reset`	Set all default values.

The controls in the **Events** area display the last events received from the viewer via the **ISTviewEvents** interface. These controls do not accept user input. Under **Display/Select** you will see the results of the **OnObjectDisplay** and **OnObjectSelect** events.

4.2.1 How the driver works

The driver is a dialog-based MFC application. It uses the **ISTview** interface to send commands to ST-Viewer, and implements the **ISTviewEvents** interface to receive events from ST-Viewer.

After the project was created with MFC AppWizard, ST-Viewer type library was used to add a class wrapper for the **ISTview** interface. ClassWizard does this using the **AddClass** button and then choosing **From a type library** on the menu. This class wrapper is usually named after the corresponding COM interface, in this application it is the **ISTview** C++ class. Then a member of this class was added to the main window class (**CVizdriverDlg**) — **CVizdriverDlg::m_iSTview**. In addition, the following code was added to the **CVizdriverApp::InitInstance** implementation:

```
if (!AfxOleInit()) {
    AfxMessageBox(IDP_OLE_INIT_FAILED);
    return FALSE;
}
```

This code initializes Automation DLLs.

The communication between the driver and ST-Viewer is established in the **CVizdriverDlg::OnInitDialog** method. The viewer is started and the new **IDispatch** object is created by the call:

```
m_iSTview.CreateDispatch("STview.Document")
```

This new **IDispatch** object is an Automation object in ST-Viewer and it impements the **ISTview** interface.

The next step is to establish return communication from the viewer to the driver via the **ISTviewEvents** interface. First, the interface is described in **vizdriver.odl** file the same way it is described in the **stview.tlb** type library:

```
[ uuid(18533554-04A8-11D3-B137-00902717498E), version(1.0) ]
library STview
{
    //  Primary dispatch interface
    [ uuid(18533554-04A8-11D3-B137-00902717498D) ]
    interface ISTviewEvents : IUnknown
    {
            [id(1)] HRESULT OnObjectDisplay(long nID,
                    long nInstance, short bDisplay);
            [id(2)] HRESULT OnObjectSelect(long nID,
                    long nInstance, short bSelect);
    };
}
```

When this ODL file is compiled, the MIDL compiler generates two C++ source files: **guid.c** and **events.h**; the former defines C++ constants with GUIDs taken from the ODL file, the latter declares the **ISTviewEvents** C++ class.

Communication becomes possible by a mechanism known as *connection points*. The **CVizdriverDlg** class is a connection point container and implements the **IConnectionPointContainer** interface. The support for this interface is provided by including an interface map and adding the appropriate interface part to it:

in **vizdriverDlg.h**:

```
DECLARE_INTERFACE_MAP()

BEGIN_INTERFACE_PART(Events, ISTviewEvents)
    STDMETHOD (OnObjectDisplay)(long nID, long nInst, short bDisp);
    STDMETHOD (OnObjectSelect) (long nID, long nInst, short bSelect);
END_INTERFACE_PART(Events)
```

in **vizdriverDlg.cpp**:

```
BEGIN_INTERFACE_MAP(CVizdriverDlg, CDialog)
    INTERFACE_PART(CVizdriverDlg, IID_ISTviewEvents, Events)
END_INTERFACE_MAP()
```

This code creates a nested class, **CVizdriverDlg::XEvents** that implements the **ISTviewEvents** interface declared in **events.h**. A member variable **XEvents m_xEvents** is added to **CVizdriverDlg** class. The **XEvents** class has two methods, **OnObjectDisplay** and **OnObjectSelect**. These methods are called when ST-Viewer fires the corresponding events.

Finally, write the implementation of **OnObjectDisplay** and **OnObjectSelect** methods (in addition to the three standard methods of **IUnknown**: **AddRef**, **Release** and **QueryInterface**). These methods are located at lines 454-501 in **vizdriverDlg.cpp**.

The connection points are initialized in the **CVizdriverDlg::OnInitDialog** method:

```
[1]   m_pEvents = m_iSTview.m_lpDispatch;

      ISTviewEvents* pEvents;
[2]   HRESULT hr = m_xEvents.QueryInterface(IID_ISTviewEvents,
             (void**)&pEvents);
      ASSERT(SUCCEEDED(hr));
      m_xEvents.Release();

[3]   if (!AfxConnectionAdvise(m_pEvents, IID_ISTviewEvents, pEvents,
           FALSE, &m_dwConnectID))
        MessageBox("AfxConnectionAdvise failed", "Error", MB_OK);
```

First, we get a pointer to the **IDispatch** interface in the viewer's Automation object [1]. This interface pointer identifies the *source* of the events. We get another interface pointer by calling **QueryInterface** on **m_xEvents** with the GUID of the **ISTviewEvents** [2]. This interface pointer identifies the *sink* or destination for the events.

Finally, the connection is esablished between the source (**m_pEvents**, a pointer to the ST-Viewer's **IDispatch**), and the sink (**pEvents**, a pointer to the driver's implementation of the **ISTviewEvents** interface) [3].

Once the bidirectional communication is set up, the driver operation is simple. Each command handler (e.g. **CVizdriverDlg::OnCommandsDisplayselectSelect()** calls the methods of **ISTview** class, which in turn invokes the interface methods in ST-Viewer. On the other end, when the viewer fires an event (that is it calls a method of **ISTviewEvents** interface), the corresponding method of **CVizdriverDlg::XEvents** is invoked.

When the driver's dialog box is closed, the **OnCancel** method calls **CVizdriver::Disconnect**, which calls **AfxConnectionUnadvise** to break the connection between the source and the sink of the **ISTviewEvents** interface.

4.3 The Visual Basic example

The window and controls of Visual Basic automation driver are identical to that of the Visual C++ driver.

4.3.1 How the driver works

The Visual Basic project for the driver consist of a single dialog form. The ST-Viewer object variable is declared at the beginning of the file as:

```
Dim WithEvents STV As STview.Document
```

The **WithEvents** keyword specifies that the **STV** object variable can respond to events by an ActiveX object (ST-Viewer automation object in this case).

For this to work, the project must reference the ST-Viewer type library (**stview.tlb**) Use the **Project | References** command in the Visual Basic editor to reference this library.

This variable is insantiated in the **Form_Load** procedure:

```
Set STV = New STview.Document
```

This insures that ST-Viewer starts and creates a new automation object as soon as the driver's dialog gets displayed.

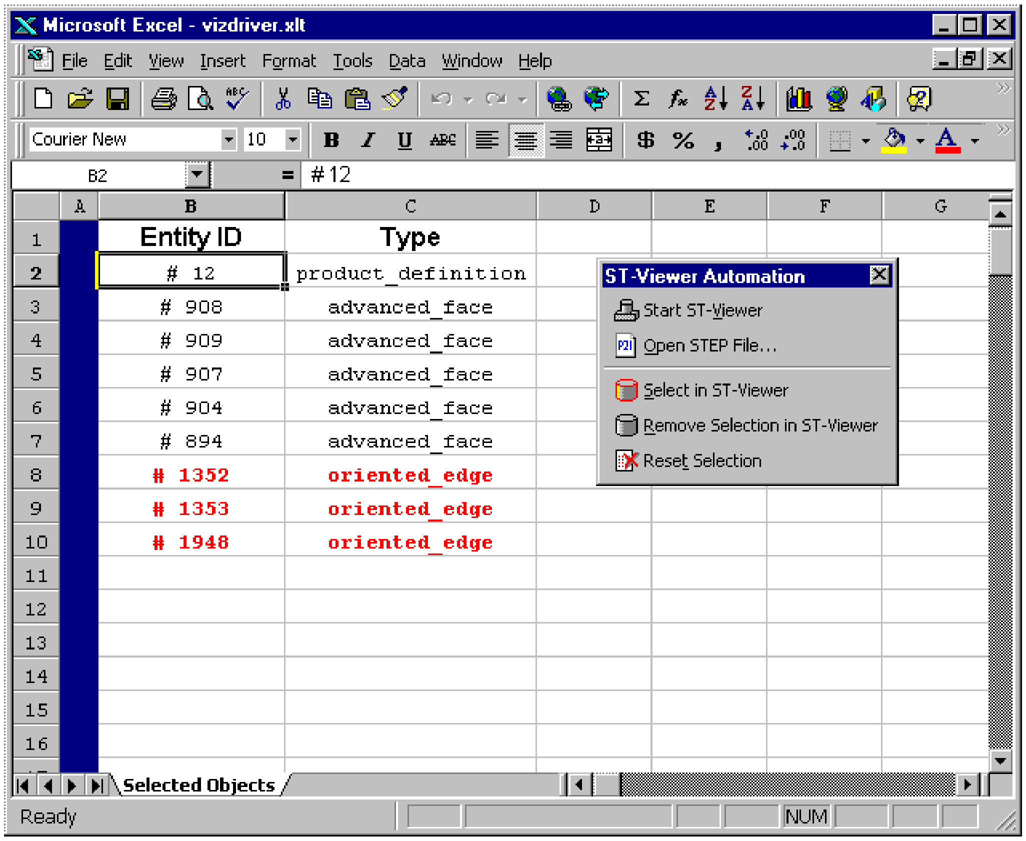
The rest of the code is very simple. Each control event procedure calls the corresponding method of the **STV** object, for instance:

```
Private Sub cmdDisplaySelectSelect_Click()
    If chkDisplaySelectAll.Value = 1 Then
        STV.MakeAllSelected (txtDisplaySelectEntityID.Text), 1
    Else
        STV.MakeSelected (txtDisplaySelectEntityID.Text),
                         (txtDisplaySelectInstance.Text), 1
    End If
End Sub
```

In addition to control event procedures, there are two event procedures for the **STV** object, **STV_OnObjectDisplay** and **STV_OnObjectSelect**. These procedures are called when the viewer fires the corresponding events (calls **ISTviewEvents** methods).

4.4 The Microsoft Excel (VBA) example

The Microsoft Excel example is a template (**.xlt**) file. This template file contains a single worksheet, **Selected Objects**. The first two columns are reserved for the selected objects IDs and types. The ST-Viewer Automation toolbar visible on the following illustration is included in the template file.



	A	B	C	D	E	F	G
1		Entity ID	Type				
2		# 12	product_definition				
3		# 908	advanced_face				
4		# 909	advanced_face				
5		# 907	advanced_face				
6		# 904	advanced_face				
7		# 894	advanced_face				
8		# 1352	oriented_edge				
9		# 1353	oriented_edge				
10		# 1948	oriented_edge				

Figure 4.2 — Microsoft Excel example window.

The file also contains a set of macros written in Visual Basic for Applications (VBA). These macros are organized into four modules within the template file.

4.4.1 How the Macros Work

The **ThisWorkbook** module defines the ST-Viewer object variable, which is declared at the beginning of the source code as:

```
Dim WithEvents STV As STview.Document
```

The **WithEvents** keyword specifies that the **STV** object variable can respond to events by an ActiveX object (ST-Viewer automation object in this case).

For this to work, the project must reference the ST-Viewer type library (**stview.tlb**) Use the **Project | References** command in the Visual Basic editor to reference this library.

The **STV** variable is insantiated by the **StartViewer** procedure in the **Viewer** module:

```
Set ThisWorkbook.STV = New STview.Document
```

The **StartViewer** procedure is called when you click **Start ST-Viewer** on the **ST-Viewer Automation** toolbar. To load a STEP file into the viewer, click **Open STEP File**. This will call the **LoadSTEPFile** procedure in the **Viewer** module. This procedure in turns calls the **STV.LoadFile** method.

When you select an object in ST-Viewer, the STEP ID of this object and its entity type are added to the worksheet and drawn in red color as shown in Figure 4.2. If the object loses selection, the text color returns to normal.

The **Main (Selected Objects)** module contains the macros that handle the rest of the commands on the **ST-Viewer Automation** toolbar — **SelectObject**, **RemoveSelection**, **ResetSelection**. These procedures call the **MakeAllSelected** method of the **STV** object.

Finally, the **ThisWorkbook** module contains the event procedures that respond to events in the viewer and in the workbook itself. The **STV_OnObjectSelect** procedure implements the **ISTviewEvents::OnObjectSelect** interface method and is called by ST-Viewer whenever the user changes the selection state of an object in the viewer. The other two procedures respond to activation and deactivation of the workbook by adding and removing the commands to control ST-Viewer from the Excel shortcut menu.