Using OLE in OpenInsight

Introduction

When Mike Ruane announced that Revelation Software was committed to releasing a 32-bit version of OpenInsight, many developers, perhaps most, were skeptical. Of course Mike and his team had done nothing personally to deserve this skepticism. They were the unfortunate recipients of a customer base that had been exposed too long and too often to false promises. To everyone's delight and mild amazement, in February 2002 the Revelation community was introduced to the first release of a 32-bit version of OpenInsight: 4.0.x.

Spurred on by their own success, Revelation seized the opportunity to explore and exploit everything they could get their hands on. They were committed to pouring as much new technology and features into OpenInsight as their imagination and resources would permit.

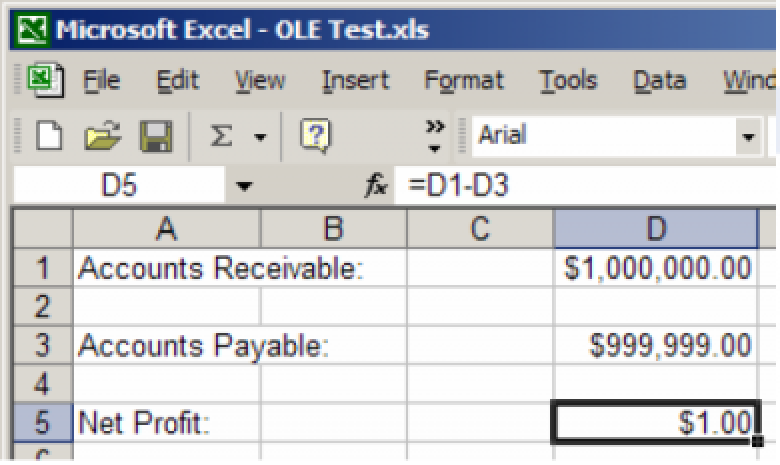
So once again the Revelation community had their expectations surpassed with another major release of OpenInsight within only six months of their previous version: 4.1.x. Many new features were added that are worth exploring: multi-instance/remote OpenEngines, Native XML support, and the OECGI. In this document, OpenInsight's support for OLE controls will be discussed.

Understanding OLE

Normally we prefer to limit our presentations to material that is directly related to OpenInsight. We recognize that many developers might be unfamiliar with terms and concepts that are typically used in other programming environments and technologies. However, since the primary purpose of OLE is to enable technologies from potentially different sources to perate as one system, it is important to review the history and theory behind OLE.

OLE is an acronym for Object Linking and Embedding. Its name implies that objects are being used *to link to or be embedded within*another application. It is a fundamental part of those Window's programs, especially productivity applications, which are designed to work together. Microsoft Office is a typical example of this.

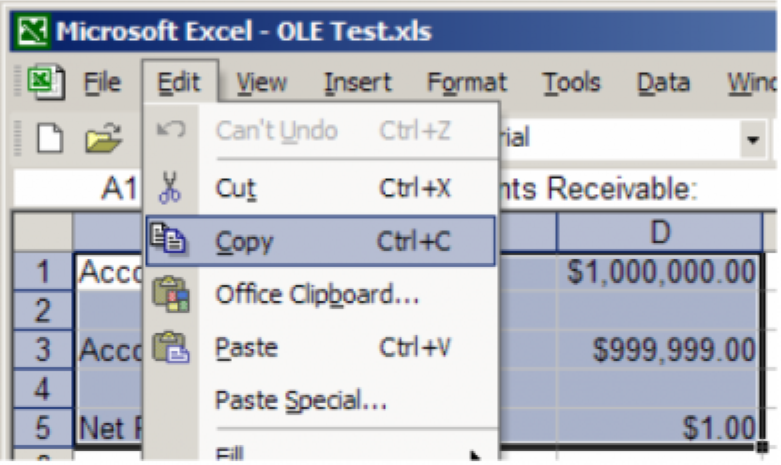
A simple illustration would be taking an Excel table and putting it within a Word document. To demonstrate, create a simple Excel document with at least one formula like this:



D5 =D1-D3

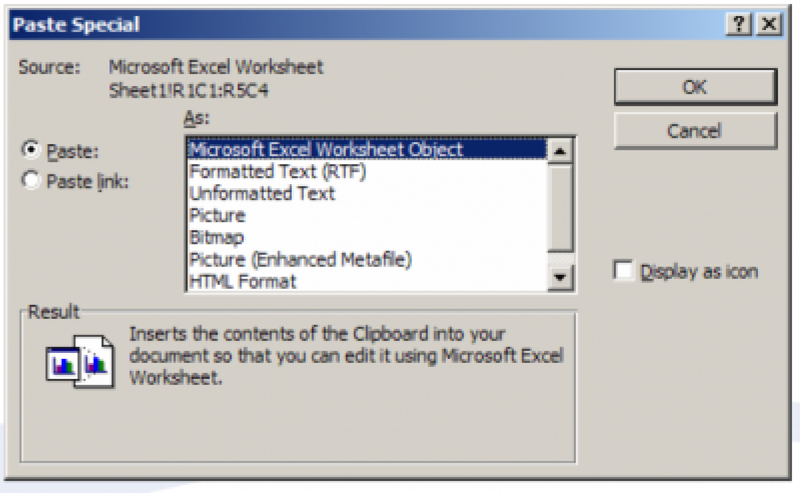
	A	B	C	D
1	Accounts Receivable:			$1,000,000.00
2				
3	Accounts Payable:			$999,999.00
4				
5	Net Profit:			$1.00

Now select cells A1 through D5 and copy them to the clipboard:



When you open up a Word document you will see two options for bringing in this clipboard data: “Paste” and “Paste Special” (see above picture for an example.) Paste does its normal job and simply creates a static table and text area. This information no longer has any connection to Excel. Even the formula we created in Excel has been converted to simple text.

Paste Special brings up a dialog box that gives us two options: “Paste” and “Paste link”. Note the descriptions provided in the Result group box based on which one is selected:



Source: Microsoft Excel Worksheet
Sheet1!R1C1:R5C4

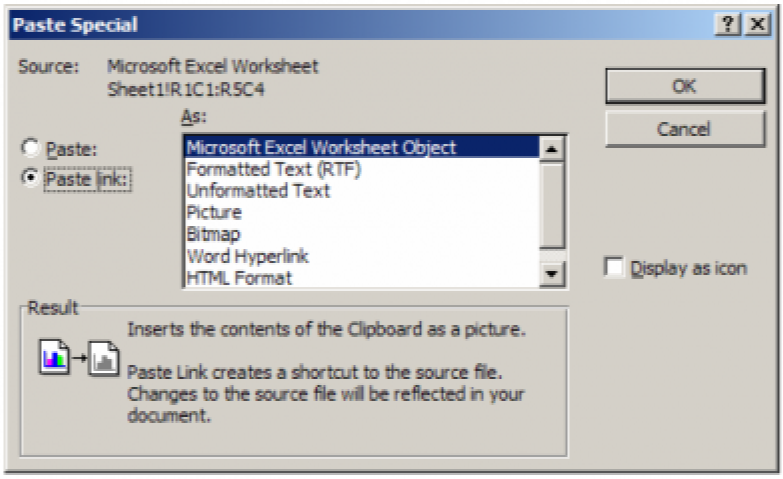
As:

● Paste:
○ Paste link:

Microsoft Excel Worksheet Object
Formatted Text (RTF)
Unformatted Text
Picture
Bitmap
Picture (Enhanced Metafile)
HTML Format

□ Display as icon

Result
Inserts the contents of the Clipboard into your document so that you can edit it using Microsoft Excel Worksheet.



Source: Microsoft Excel Worksheet
Sheet1!R1C1:R5C4

As:

○ Paste:
● Paste link:

Microsoft Excel Worksheet Object
Formatted Text (RTF)
Unformatted Text
Picture
Bitmap
Word Hyperlink
HTML Format

□ Display as icon

Result
Inserts the contents of the Clipboard as a picture.

Paste Link creates a shortcut to the source file. Changes to the source file will be reflected in your document.

To summarize, *Paste Special* -> Paste brings a copy of our Excel table (including formulas) into Word while maintaining its property as an Excel table. Therefore, changes to the table data can be made directly in Word and any formulas will re-calculate as needed. This is an example *Object Embedding*.

Paste Special -> Paste link simply brings a visual display of our Excel table into Word but the source of this data is still within our original Excel document. So when changes are made to the Excel document we can see the results displayed in Word. This is an example of Object *Linking* and in many ways is similar to the concept of symbolic fields in OpenInsight.

Conceptually, OLE represents the evolution of object-oriented programming that was introduced through the use of DLLs and, in some ways, DDE (Dynamic Data Exchange). Both of these older technologies have been supported by OpenInsight from its inception. However, OLE is easier to reuse and share because it is based on design specifications that require standard interface support and backwards compatibility. Whereas changes to a DLL might break an application, changes to an OLE object should not affect any systems built around an older version. Anyone who has installed a commercial application using a newer version of datatbl.dll has experienced how the edittable control can start behaving differently than expected.

Perhaps the most common use of OLE is through ActiveX controls; also known has OCX because of the .OCX extension of their system files. ActiveX provides very similar functionality as DLLs in that they contain callable stored procedures (known as “methods”) and they provide a user interface. Hence, when an ActiveX control is placed within a container that can support this type of OLE, e.g. an OpenInsight form, the end user is presented with its pre-designed visual interface. While ActiveX controls can be designed as very complicated self-contained applications, they are normally designed to fulfill relatively simple user interface needs. Common examples include: command buttons, picture viewers, tab bars, and spin controls.

OLE in OpenInsight

Revelation has metaphorically put their “foot in the door” with their current support for OLE. While there are wonderful advantages and features that can be utilized right now, OLE implementation is somewhat rough and limited. Revelation has expressed their intention to improve OLE support but we probably won’t see anything significant until version 7.1 or later. Therefore, it is important to know what *can* and *cannot* be done now with OLE in OpenInsight.

At present OpenInsight only supports ActiveX/OCX. Consequently, for the remainder of this document, unless otherwise noted, the terms ActiveX, OCX, and OLE will have synonymous meanings. Even though ActiveX is not as sophisticated as linking/embedding productivity applications, it does provide developers an opportunity to enhance their OpenInsight applications (visually and functionally) in a virtually unlimited number of ways.

Programmers who are familiar with other visual programming tools (like Delphi, Visual Basic, etc.) know that the OpenInsight control palette provides very limited options. For instance, one of the most common user interface objects in any Windows application is the tab control. Since tabs are not integrated into the Form Designer, developers have to write complicated code and/or spend a lot of time designing bitmaps to simulate a working tab control.

OLE via ActiveX is a great way to overcome this kind of limitation without having to upgrade the Form Designer whenever a new control is desired. Adding a new control to OpenInsight now only requires the installation of an ActiveX control of one's choice into Windows, placing an OLE object on a form, and then providing the necessarily information to link the OLE object to the desired ActiveX control.

Getting Started

There are a few items that are needed to successfully use OLE controls in OpenInsight. Each of these will be discussed in greater detail:

1. One or more controls with the .OCX extension (e.g. MSCAL.OCX)
2. Current and complete documentation on each control's properties, events, and methods
3. Each OCX control's ProgID or CLSID

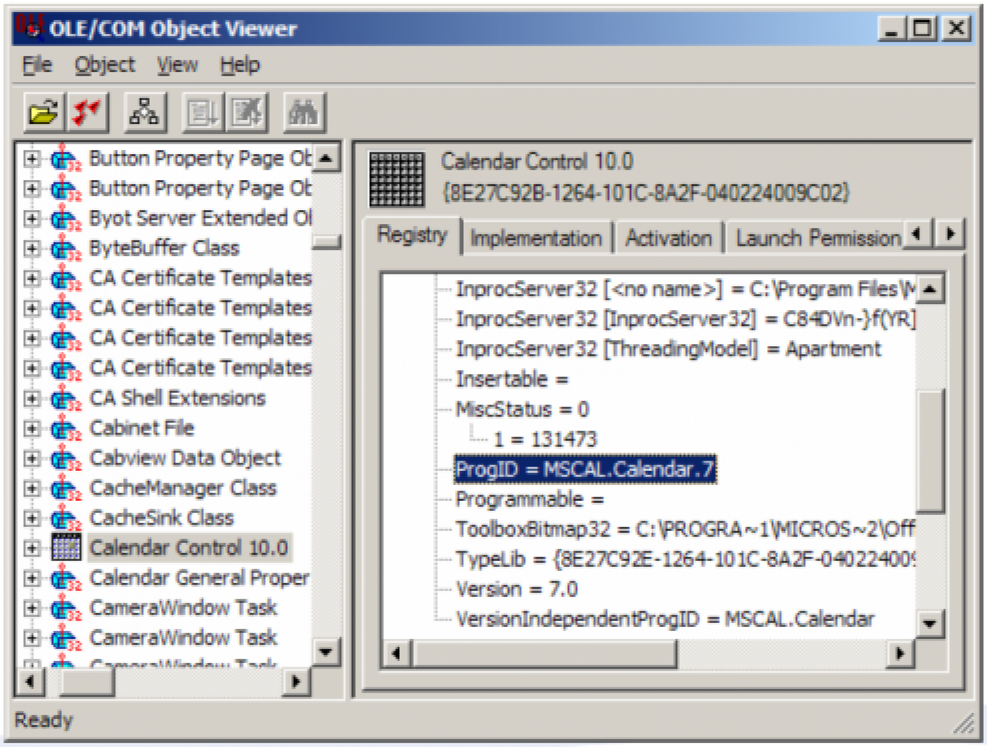
There are three basic ways to acquire OCX controls: 1.) Use ones that are already supplied with the Windows operating system, 2.) acquire commercial, shareware, or freeware controls from a third-party (usually through the internet), or 3.) write your own. Since most of us will be unable to write our own we need to know what to look for with third-party or Windows supplied controls.

If you ever do an online search for OLE controls (e.g. "free activex", "free ocx", "ole controls", etc.) you will find more links than you could possibly exhaust. What will become obvious rather quickly is that the majority of available controls are designed primarily for Visual Basic. While this doesn't preclude other development environments from using these OLE controls, it is likely that some of the functionality will be limited. This is because people who design controls from within Visual Basic for Visual Basic depend upon commands that are unique to Visual Basic. That is to say, they were not interested in having their controls utilized in other environments (like Delphi, OpenInsight, etc.) and therefore they took advantage of pre-built Visual Basic features. This makes their work easier but unfortunately it makes their controls somewhat limited. Therefore we suggest that you look for controls that are designed to be friendly to all development platforms. Usually anything designed with Visual C++ or Delphi will work fine.

Something that we have seen lacking in share/freeware downloads is sufficient documentation. Typically, controls are bundled with a sample Visual Basic application instead. So unless you have access and understanding of Visual Basic this will be useless (note: Visual Basic applications usually come with . FRM files. If you edit this with a simple text editor you will see how the form stores the initial property settings for its ActiveX controls. This information is pretty easy to interpret and can be a way of identifying control properties and supported values.) Do everything possible to acquire complete documentation; your life will be so much easier.

Finally, in order for OpenInsight to incorporate an OCX control into the form, we must provide it the control's ProgID or CSID. Hopefully this information will be provided in the documentation. If not, then we need to search for these ourselves.

One helpful tool is Microsoft's OLE/COM Object Viewer (see the References section below). However, we have not found any easy way to copy and paste the information it provides. Here is an example of what this tool will show you when examining the Microsoft Calendar Control (MSCAL.OCX):

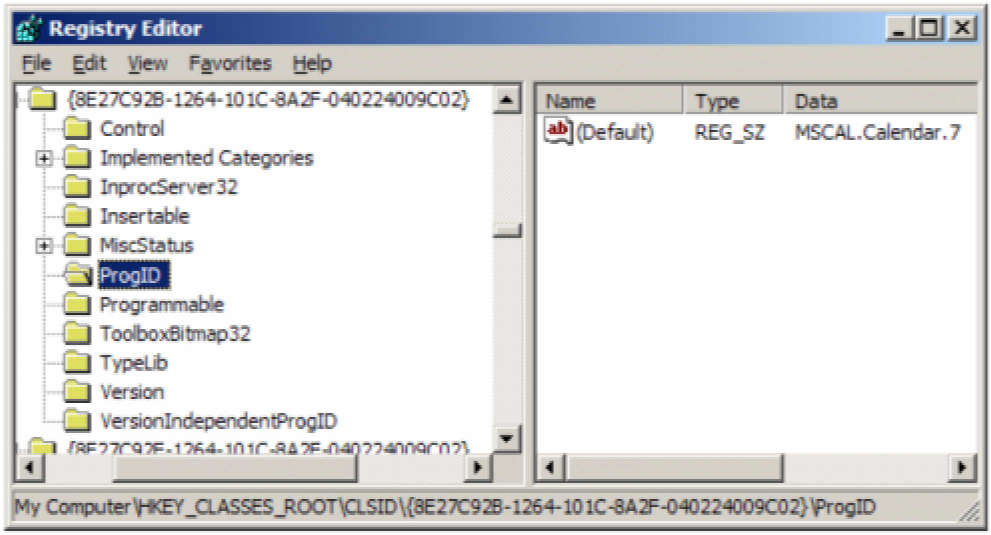


Calendar Control 10.0
{8E27C92B-1264-101C-8A2F-040224009C02}

```
InprocServer32 [<no name>] = C:\Program Files\M
InprocServer32 [InprocServer32] = C84DVn-}f(YR]
InprocServer32 [ThreadingModel] = Apartment
Insertable =
MiscStatus = 0
    1 = 131473
ProgID = MSCAL.Calendar.7
Programmable =
ToolboxBitmap32 = C:\PROGRA~1\MICROS~2\Off
TypeLib = {8E27C92E-1264-101C-8A2F-04022400
Version = 7.0
VersionIndependentProgID = MSCAL.Calendar
```

Note that the ProgID line has been highlighted. This tells us that MSCAL.Calendar.7 is this ActiveX control's Program Identifier (ProgID). We will need this in order to associate our OLE container object in the Form Designer to this particular ActiveX control. (Note: Five lines below the ProgID is the VersionIndependentProgID line. We can also use this value. Its purpose, as should be evident by the name, is to provide us a Program Identifier that is common to all versions of this ActiveX control.)

Another tool than can be used fairly easily is the Registry Editor. Usually you can do a search for your ActiveX control (e.g. MSCAL.OCX) and within the first few matches it should place you in the vicinity where you can retrieve the ProgID:



Name	Type	Data
(Default)	REG_SZ	MSCAL.Calendar.7

My Computer\HKEY_CLASSES_ROOT\CLSID\{8E27C92B-1264-101C-8A2F-040224009C02}\ProgID

After you have located your ActiveX control's ProgID Key, you can double-click on the value to the right to easily copy and paste to the clipboard.

An alternative to using the ProgID would be the CLSID (Class ID). This value can be retrieved from both tools that have been discussed here. In the above screen shot of the Registry Editor, the CLSID is the upper left string of numbers and letters contained within the "french" braces:

```
{8E27C92B-1264-101C-8A2F-040224009C02}
```

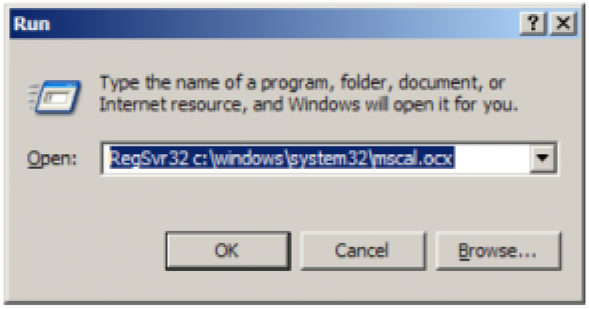
For obvious reasons we recommend using the ProgID or VersionIndependentProgID whenever possible.

ActiveX controls must be registered to the local Windows operating system in order to work. Controls that are already included with Windows are normally pre-registered. Third-party controls might do this for you during an installation process. Otherwise, you are responsible for registering them yourself as you would for any controls that you develop in-house. However, since registering a control requires that you know the path where the controls resides, a decision must first be made as to where to place any third-party or custom developed controls.

Standard practice dictates that ActiveX controls, like system DLLs, be kept in the Window's System32 folder. This provides the advantage of maintaining all controls in one place. All applications that use this control will always be using the same (and hopefully newest) version of the ActiveX control. Since the System32 folder never moves, applications can be moved at will and they will always know where to access any needed ActiveX controls.

One caveat, however, is that with large network environments any major changes to an ActiveX control will require an update to every user workstations that accesses this application. Without special tools that administrate client environments automatically, this can be a significant chore. This is particularly the case with custom developed ActiveX controls which tend to go through significant modifications more often. In this scenario, it might be easier to store the ActiveX control in the same folder where the application is. If an application's path does not change, then this can be a better solution.

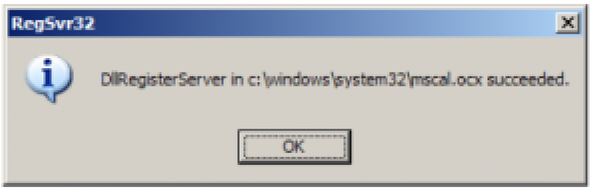
Once the path is known, the command to register a control is: RegSvr32 *pathname\controlname.ocx*. For example:



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: RegSvr32 c:\windows\system32\mscal.ocx

If all went well then you will receive a message similar to this:

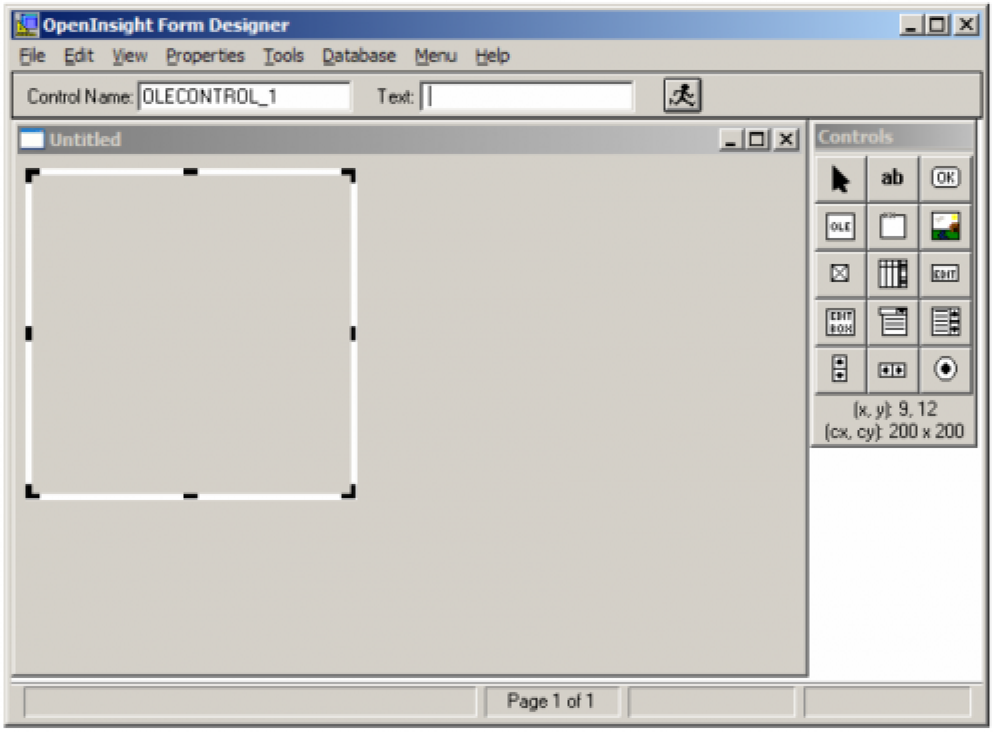


DllRegisterServer in c:\windows\system32\mscal.ocx succeeded.

If you didn’t get this message then double-check your path, spelling, and whether or not the OCX file actually exists in your system.

Putting It All Together

Once the above steps have been completed, displaying an ActiveX control in an OpenInsight form is very easy. We’ll continue to use the Microsoft Calendar as an example. Begin by creating a new form with no table. Select the OLE button from the Control palette and then click on the form to create a new OLE container object:



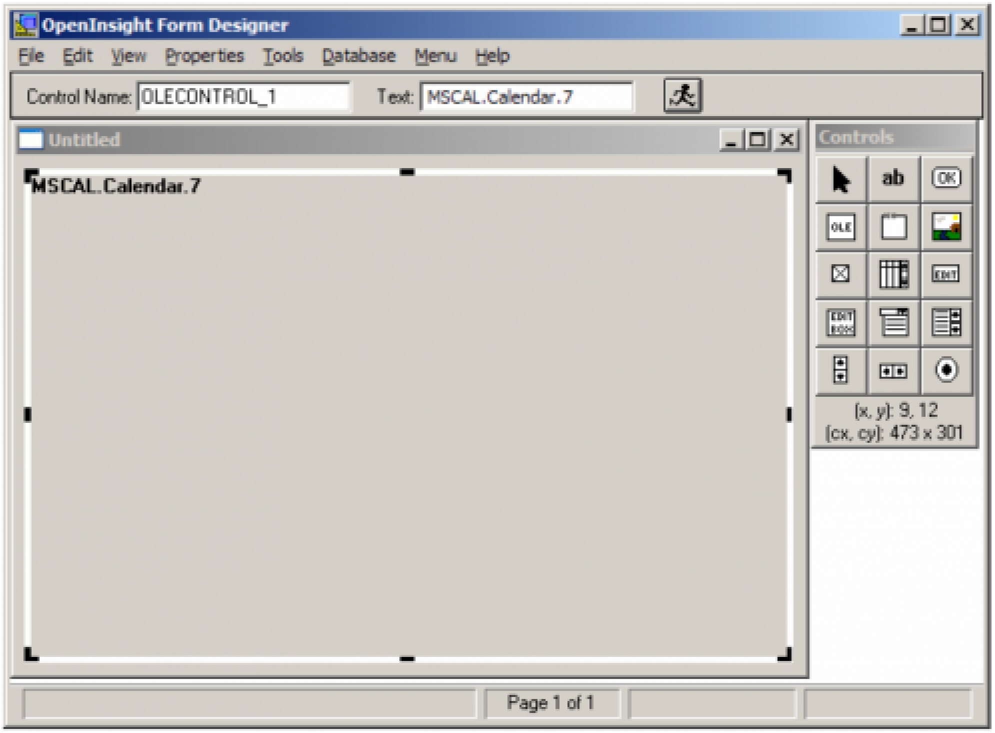
Control Name: OLECONTROL_1

Text:

(x, y): 9, 12
(cx, cy): 200 x 200

At this point it is important to understand that there are two components that have to be considered when working with OLE: the OpenInsight OLE control (which is what we just placed on our form and will act as a container) and actual ActiveX control that this container will reference. This distinction will be relevant when the subject of Properties and Events are discussed below.

Now we need to associate our ActiveX control to the OpenInsight OLE container. In the Text property field enter the ProgID (or CLSID) of the control you want. Since the Microsoft Calendar ActiveX control employs a large interface, stretch out the container to accommodate. Your window should look something like this:


OpenInsight Form Designer
File Edit View Properties Tools Database Menu Help
Untitled
Controls
ab
OK
OLE
EDIT
EDIT BOX
Page 1 of 1

Control Name: OLECONTROL_1

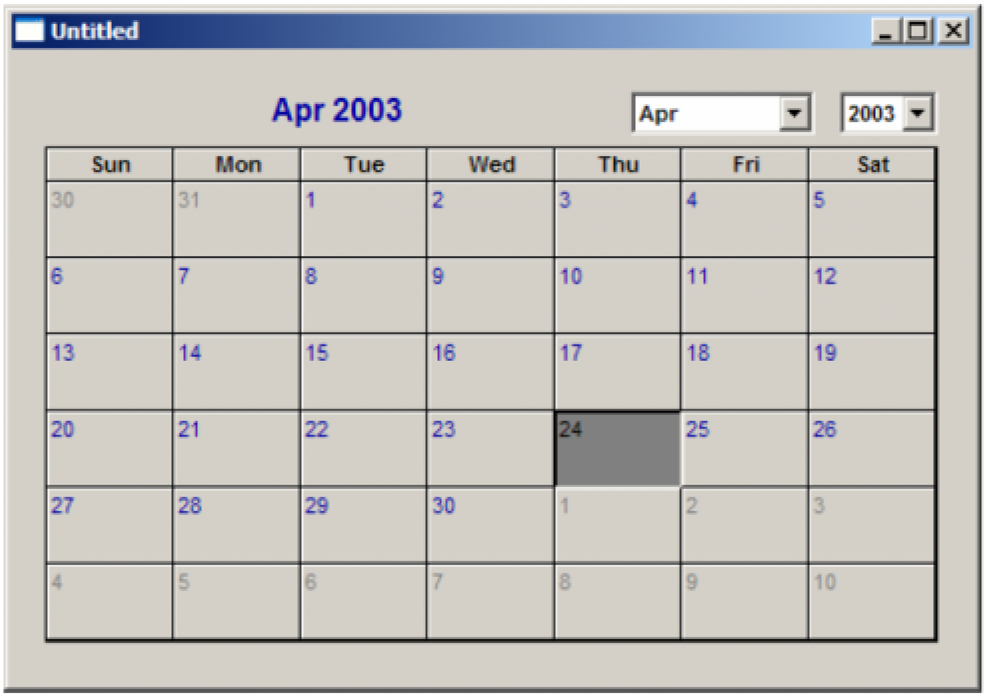
Text: MSCAL.Calendar.7

MSCAL.Calendar.7

(x, y): 9, 12

(cx, cy): 473 x 301

Go ahead and test run your form, naming it OLE_TEST. It should look like this:



Apr 2003

Apr ▾ 2003 ▾

Sun	Mon	Tue	Wed	Thu	Fri	Sat
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

In one sense, that's all there is to it! This calendar control can now respond to user interaction. Different dates can be selected and the month or year can be changed at will. No Basic+ programming was required to enable this functionality. Remember, ActiveX controls are essentially self-contained programs that only require an environment to operate within. Therefore, any built-in functionality will immediately be available to the end-user without any intervention by the OpenInsight developer.

While the ActiveX control can certainly operate without additional assistance from OpenInsight, it will need help to communicate and interact directly with the development environment. For instance, even though the end user can easily change the date on the calendar, the ActiveX control hasn't told OpenInsight what this date is. So even though the end user can see the date on the screen, the application itself needs to know this date if it is going to use it in a program or store it in the database.

Since OLE uses communication concepts that we are already familiar with (i.e. Properties, Events, and Stored Procedures (normally called Methods)) we can leverage this knowledge without too much of a learning curve. Additionally, Revelation Software has very nicely adapted the Set_Property(), Get_Property(), and Send_Message() functions to work with OLE controls. This is good news for those who are concerned about having to learn a whole new tool set in order to use ActiveX in their OpenInsight applications.

Properties

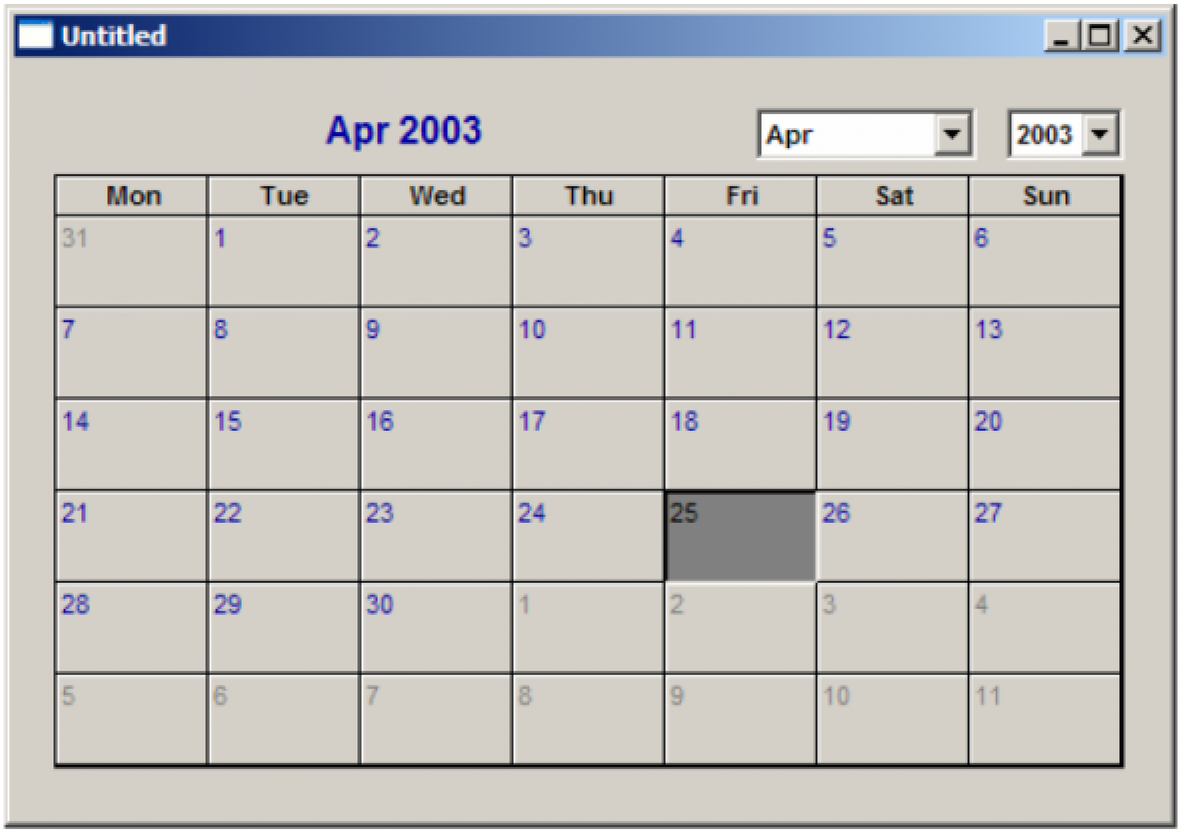
Just like native OpenInsight controls, OLE controls have properties that are set or retrieved. Note, however, that in the case of OLE controls there are the properties of the OpenInsight OLE container (which are regular native OpenInsight properties) and there are properties that are inherent to the OLE (i.e. ActiveX) control itself. As we will shortly see, there needs to be a way to distinguish between them.

If we wanted to change the text that appears in a standard editline we would do something like this:

```
rv = Set_Property(@Window:".EDITLINE_1", "TEXT", "Hello World!")
```

We can use the exact same logic to modify the property of an OLE control. For instance, the Microsoft Calendar control has a property called FirstDay which tells the control which day of the week should be displayed in the first column (see References section for a complete list of properties, methods, and events for the Microsoft Calendar control.) Let's say we wanted to create a calendar that is oriented toward a business schedule (similar to how Day Runner calendars are formatted). We would want Monday to appear in the first column. This can be done by placing this logic in the CREATE event then test running the form:

```
rv = Set_Property(@Window:".OLECONTROL_1", "FirstDay", 1) ; * 1 = Mon, 2 = Tues, etc.
```



Apr 2003

Apr

2003

Mon	Tue	Wed	Thu	Fri	Sat	Sun
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

Notice, however, that the Microsoft Calendar property is in mixed case. This is generally the way ActiveX control properties are named. It is imperative that the correct case be used or they won't be recognized correctly. Which brings us to an interesting question: what if the property in the ActiveX control has the same name and case as the native OpenInsight property?

To resolve this potential issue, Revelation has provided for the keyword "OLE" to be placed before the property name using a dot separator. For instance:

```
rv = Set_Property(@Window:".OLECONTROL_1", "OLE.FirstDay", 1)
```

This will force OpenInsight to treat this property as specific to the ActiveX control rather than for the OpenInsight OLE container itself. Granted, since OpenInsight properties are always uppercase it would seem unlikely that there would be duplicate property names. Regardless, SRP has chosen to always use the “OLE” keyword since this can also serve as a visual indicator that the control being referenced is an ActiveX control rather than a native OpenInsight control.

Events

So how does the ActiveX control notify OpenInsight when the end user performed an action? In our current example, how can we detect when the end user changed the date?

Fortunately this is done in exactly the same way OpenInsight handles it, through events (actually, to be more accurate, OpenInsight and OLE both work the way the Windows operating system was designed:*event driven*.) Depending on how the ActiveX control was designed, an event is triggered when certain end user actions are performed. In the case of the Microsoft Calendar control, there are a few that can be useful to us (again, see the Reference section for a complete list):

AfterUpdate	Occurs after the user selects a new date in the control and the new date is highlighted.
Click	Occurs when the user clicks on a date in the control.
DblClick	Occurs when the user double-clicks on a date in the control.

Usually we don’t care when the end user is changing dates. We only want to know when they are done. With the Calendar control this can be done in two ways: 1.) Place an OK button on the form, and/or 2.) use the DblClick event has an indicator that the end user has selected their date. If we follow good and intuitive user interface design practices we would do both! For simplicity in this document, however, we will use the DblClick event.

Even though an ActiveX control has its own events that will launch when required, we need to tell OpenInsight to do two things: 1.) What OLE event(s) to listen for, and 2.) what do with these events when they are heard. Both requirements are handled through one Send_Message() command.

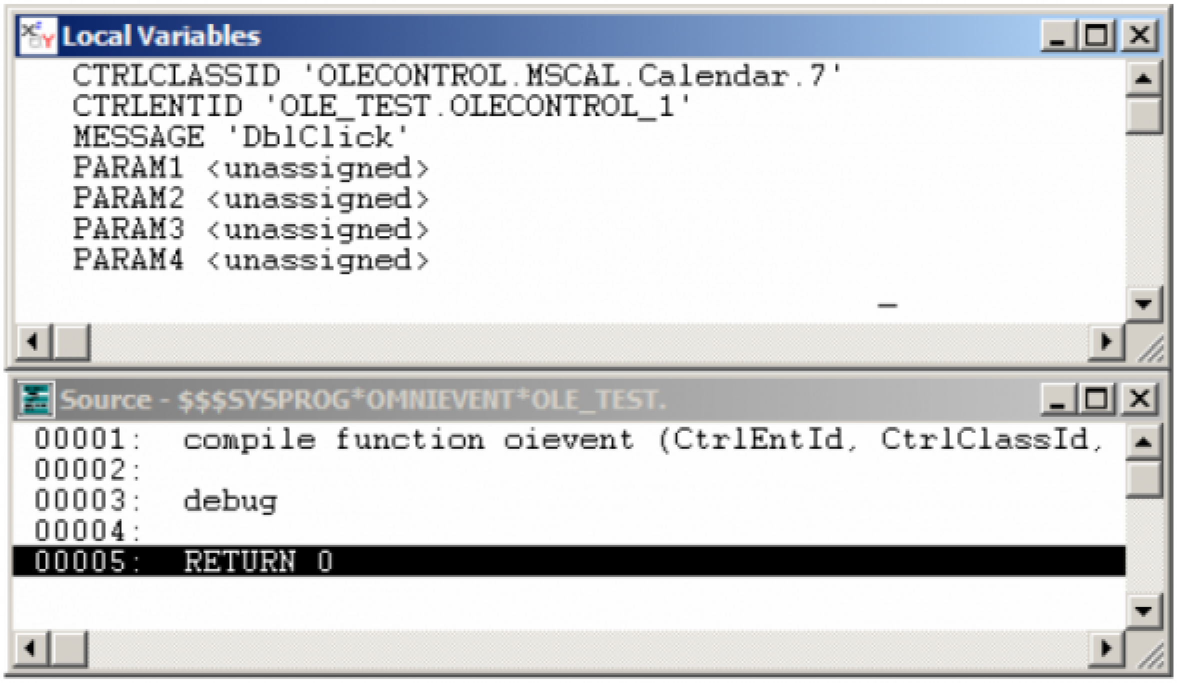
If you have experience with using the Send_Message() function to redirect events (via the QUALIFY_EVENT message) then this will be very familiar. If not, consult the Programmer’s Reference Guide and then download our Creating a Custom Event white paper for a fuller treatment on the subject. Consider the following statement:

```
rv = Send_Message(@Window:".OLECONTROL_1", "QUALIFY_EVENT", "DblClick", 1)
```

By placing this code in an appropriate place, e.g. the CREATE event of our window, we are requesting that OpenInsight be notified when the DblClick event of our ActiveX control fires. By default, the OLE event of our OLE container will receive the event call. Both the script and QuickEvent handlers are available to the developer. However, a developer might want to capture all of the OLE events in one place (i.e. using the concept of a commuter module). To do so we can amend the *eventprocessing* parameter in our Send_Message() statement to specify an alternative event. In our example we will direct our ActiveX events to the OMNIEVENT of our window:

```
rv = Send_Message(@Window:".OLECONTROL_1", "QUALIFY_EVENT", "DblClick", 1:@FM:
-> "5*":@AppId<1>:"*OMNIEVENT*OLE_TEST.")
```

Now the OpenInsight programmer can respond as needed. If we put a debug statement in our OMNIEVENT, run our window, and then double-click on a date, this is what we’ll see:



```
CTRLCLASSID 'OLECONTROL.MSCAL.Calendar.7'
CTRLENTID 'OLE_TEST.OLECONTROL_1'
MESSAGE 'DblClick'
PARAM1 <unassigned>
PARAM2 <unassigned>
PARAM3 <unassigned>
PARAM4 <unassigned>
```

```
compile function oievent (CtrlEntId, CtrlClassId,

debug

RETURN 0
```

Consequently, we have enough information to determine exactly what is happening with our ActiveX control: the type of ActiveX being used, the name of the OLE control on our OpenInsight form, and the name of the ActiveX event that was triggered. (Note: ActiveX events can also support additional parameters. These will be passed through our additional OpenInsight event parameters in sequence. Because the OMNIEVENT only provides for a total of five additional parameters, SRP has created a promoted OLE event to manage all ActiveX event processing since it allows for up to eleven additional parameters.)

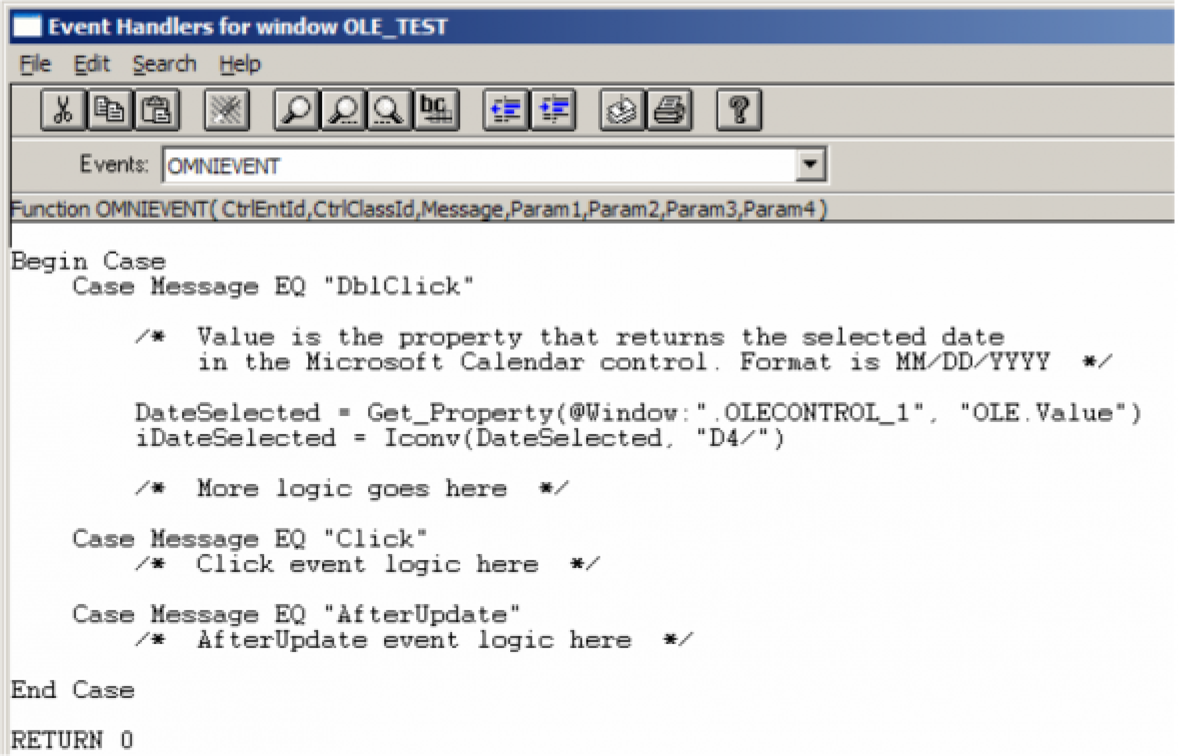
Event names are also case sensitive. Therefore, as with properties, we can use the keyword “OLE” with the dot separator in our name to make a clear distinction between OpenInsight events and the ActiveX events:

```
rv = Send_Message(@Window:".OLECONTROL_1", "QUALIFY_EVENT", "OLE.DblClick", 1:@FM:
-> "5*":@AppId<1>:"*OMNIEVENT*OLE_TEST.")
```

One very helpful feature that Revelation has added is another special keyword: “ALL_OLES”:

```
rv = Send_Message(@Window:".OLECONTROL_1", "QUALIFY_EVENT", "ALL_OLES", 1:@FM:
-> "5*":@AppId<1>:"*OMNIEVENT*OLE_TEST.")
```

By using this we are requesting that all events for our ActiveX control be re-directed and in our example above, they will all be directed to the OMNIEVENT of our window. This eliminates the need to write a Send_Message() line of code for every ActiveX event and it can help us figure out what events exist in our ActiveX controls when our documentation is incomplete. Since all necessary information is passed through the OpenInsight event parameters, we can use case logic to react accordingly. So now we can add an ActiveX property check in our event logic to complete our goal of having OpenInsight made aware of the date being selected (via a double-click):



```
Function OMNIEVENT( CtrlEntId,CtrlClassId,Message,Param1,Param2,Param3,Param4 )

Begin Case
    Case Message EQ "DblClick"

        /*  Value is the property that returns the selected date
            in the Microsoft Calendar control. Format is MM/DD/YYYY   */

        DateSelected = Get_Property(@Window:".OLECONTROL_1", "OLE.Value")
        iDateSelected = Iconv(DateSelected, "D4/")

        /*  More logic goes here  */

    Case Message EQ "Click"
        /*  Click event logic here  */

    Case Message EQ "AfterUpdate"
        /*  AfterUpdate event logic here  */

End Case

RETURN 0
```

By combining OLE events and properties we are now able to interact with an ActiveX control and use them relatively seamlessly in our application. All in all, this is not a difficult endeavor.

Methods (aka Stored Procedures)

ActiveX controls can also contain callable stored procedures, normally referred to as methods. Usually these are meant to enable the development environment a means for controlling the ActiveX control through its own user interface controls, like a pushbutton. However, this is by no means the only purpose for methods.

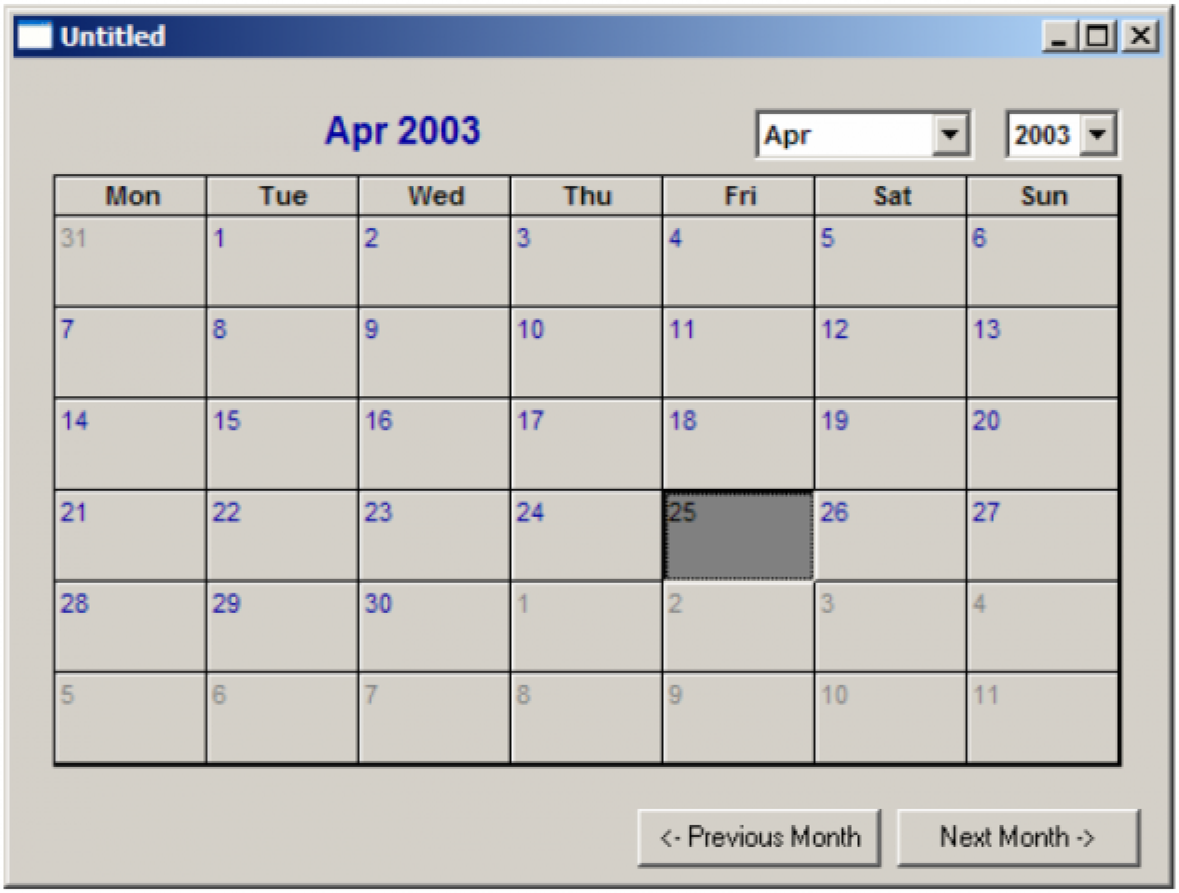
Methods are called by using the Send_Message() function. In the message parameter we'll put the name of the method we want to call (again, we can precede this with the "OLE" keyword like we can with properties and events.) Method parameters, if any, can simply be passed through the available param1 through param4 parameters:

```
rv = Send_Message(@Window:".OLECONTROL_1", "OLE.Method", Param1, Param2, . . .)
```

If the method uses more than four parameters then we must append the message parameter with an @FM and a 1. Then all of the method parameters have to be sent as an @FM delimited list in param1:

```
rv = Send_Message(@Window:".OLECONTROL_1", "OLE.Method":@FM:1, Param1:@FM:Param2 . . .)
```

Now let's complete our sample form by adding two pushbuttons underneath the OLE control that will be used to advance or back up the calendar one month at a time:



Mon	Tue	Wed	Thu	Fri	Sat	Sun
31	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	1	2	3	4
5	6	7	8	9	10	11

In the CLICK events for each button put the corresponding line of code:

```
rv = Send_Message(@Window:".OLECONTROL_1", "OLE.PreviousMonth")

rv = Send_Message(@Window:".OLECONTROL_1", "OLE.NextMonth")
```

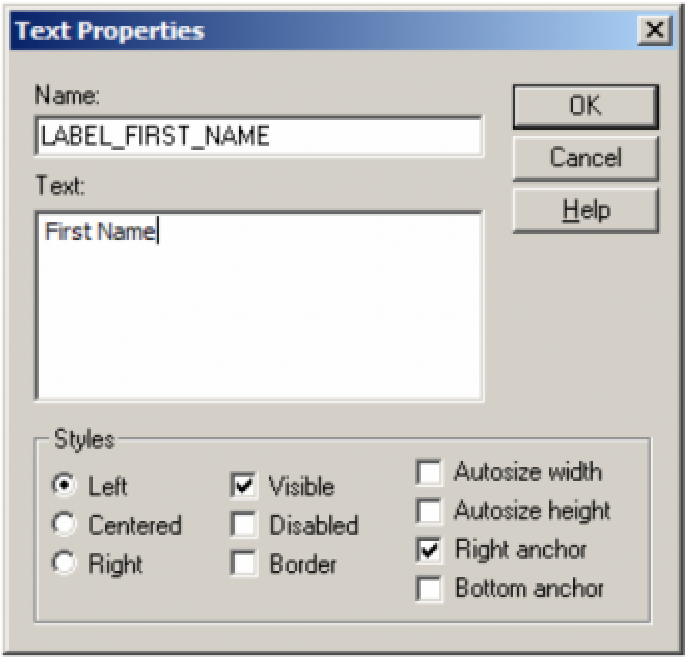
Just compile and test run the window. Voila! You have successfully added an OLE control to an OpenInsight window while integrating properties, events, and methods for complete functionality. With that, we now encourage you to explore the world of OLE to begin adding features that might never have been considered with previous versions of OpenInsight.

Limitations of OpenInsight’s OLE

Now that you have “put it all together”, you will want to be aware of some (temporary) limitations to OpenInsight’s implementation of ActiveX. To prepare you we will list them here.

1. Lack of a Design-Time Property Dialog Box

Property Dialog Boxes give the developer a way to easily specify the initial display properties of a control without having to write code. For example, the static text control uses this dialog:



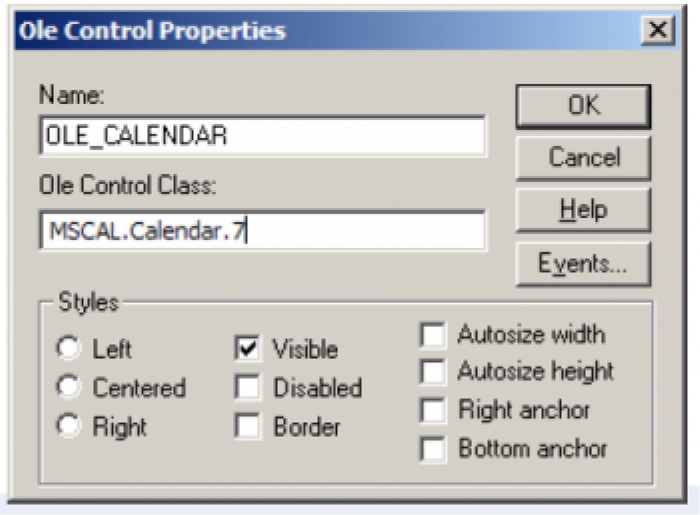
Name: LABEL_FIRST_NAME

Text: First Name

Styles

● Left
○ Centered
○ Right
☑ Visible
□ Disabled
□ Border
□ Autosize width
□ Autosize height
☑ Right anchor
□ Bottom anchor

We see that the TEXT, VISIBLE, and ENABLED properties (among others) can be easily set without a line of code. OLE container controls, however, have only this dialog:



Name: OLE_CALENDAR

Ole Control Class: MSCAL.Calendar.7

Styles

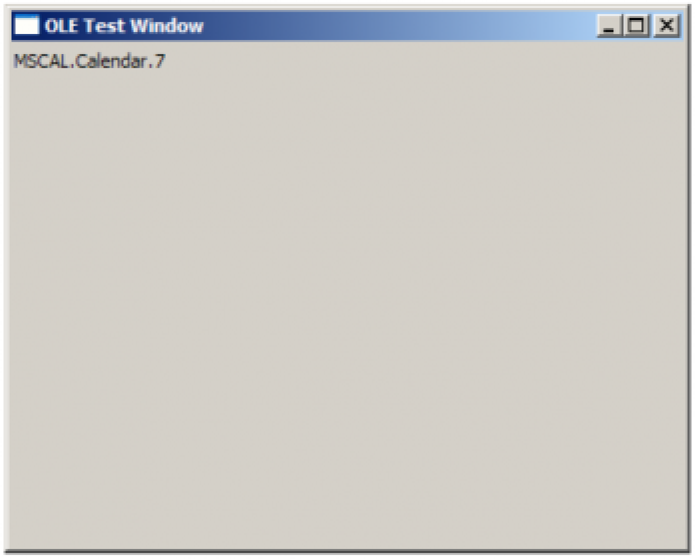
○ Left
○ Centered
○ Right
☑ Visible
□ Disabled
□ Border
□ Autosize width
□ Autosize height
□ Right anchor
□ Bottom anchor

Although there appears to be a few properties in this dialog box, these only affect the OLE container object that OpenInsight provides and not the ActiveX control that it will eventually contain. Therefore, all property settings, initial or otherwise, must be handled via code or through a tool that does the work for you (SRP has designed their own OLE Wizard to manage most of this type of work.)

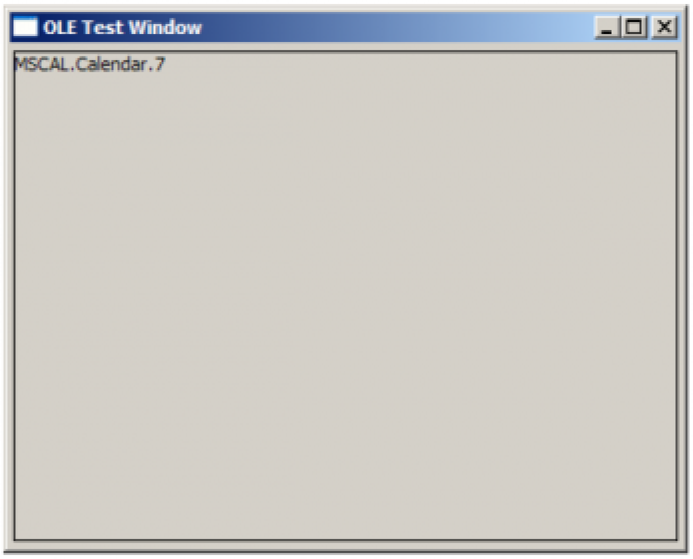
2. No Data-binding Support

Certain controls, like tabs and command buttons, would rarely, if ever, need to be tied to the database. Edit controls, however, would greatly benefit if this were possible. Consider the advantages of having a fully featured rich-text control whose contents could be edited like a word processor and could also directly read and write its information, including text formatting, to the database. Without the ability to bind OLE controls to the database, we are only half-way there. Obviously we can simulate the second half, but this requires more code.

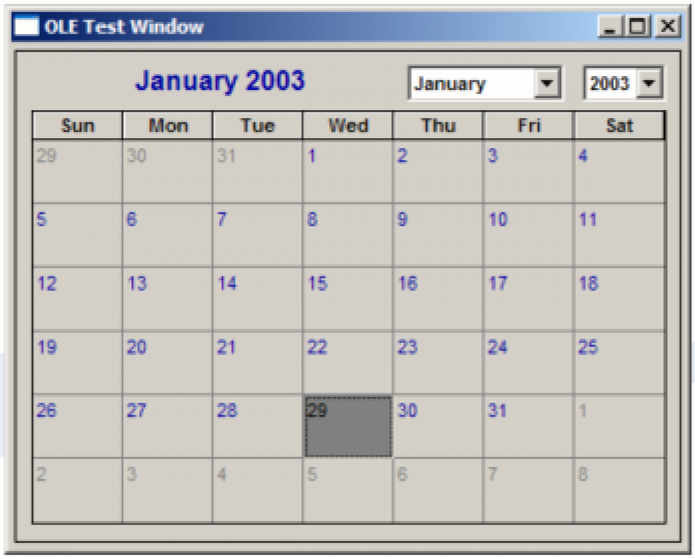
3. Implementation Only at Runtime When in the Form Designer we can only specify the name of ActiveX control we want to display in the OLE control area. Unfortunately this gives us a very limited idea of how the control will appear on the form until it is executed:



Unless we select the control (which exposes the selection handles) we have no idea how tall or wide our control is. One technique that can be used to help is to select the Border checkbox in the OLE property dialog box. This will then draw a border on the outer edge of our OLE control to make it easier to identify:



Unless you like the border to display during runtime, make sure you turn off the checkbox when the control is in its final location and size. Unfortunately we still can’t see how the control will appear to the end user until we run the window ourselves:



Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Our inability to see a visual layout becomes a very noticeable problem when several OLE controls need to be placed next to each other, like a row of buttons. Therefore, we developed within our OLE Wizard the ability to reposition OLE controls on a running window and save any changes back to the source record in SYSREPOSWINS table.

4. Limited Interaction with the OpenInsight Form and Native Controls

There are a few issues that we will identify here. First, if you are using OpenInsight v4.1.x, OLE controls will not be within the tabbing order of native controls unless you use the NEXT and/or PREVIOUS properties to modify the tab flow to include the OLE Container control. OpenInsight 7.0.x, however, allows the developer to include OLE controls in the tab order by using the Tools -> Order Tabs menu item. In either case, when the OLE control needs to lose focus the ActiveX control needs to send a TAB character via the WM_CHAR Windows message to the parent form. Therefore, third-party controls are likely to not work “as is”. Second, OLE controls prevent access to any native OpenInsight controls if they occupy the same space on the form. Normally one wouldn’t do this on purpose, but some ActiveX controls are designed to be a container for other controls (similar to the Group Box). Therefore one might attempt to do this in OpenInsight and wonder why they can’t click on their native controls. Third, OpenInsight forms behave strangely when there are only OLE controls on a form (or page). This can be resolved if the ActiveX control sends a WM_SETFOCUS Windows message to the OLE Container control (not the parent form) whenever they get focus. Alternatively, this can also be resolved by making sure at least one native control, one that can get focus, exists on every form or page that has an OLE control. If this is done, multi-page forms should explicitly set focus to a native control whenever the page is changed. This is because OpenInsight remembers the last native control that had focus. Therefore, if something triggers the form to find a control (like toggling to another window and back) we want it to focus on a control that appears on the same page. Finally, in OpenInsight v4.1.x, keyboard and mouse messages are suppressed and are not received by the parent form when an OLE control has focus. This causes features like menu accelerator keys to fail. There is a workaround, but it has to be designed within the ActiveX control itself. This will unlikely exist in any pre-built control (commercial or share/freeware), but the fix is very simple: Get the handle for the parent window of the OLE control (i.e. the OpenInsight form) and issue a command like *PostMessage(hWnd, message, wParam, lParam) for all keyboard and mouse messages*. OpenInsight 7.0.x resolves this problem and does not require any additional help from the ActiveX control.

References

Further information we think will be helpful are listed here for your convenience:

1. Microsoft’s OLE/COM Object Viewer can be downloaded here:

http://www.microsoft.com/com/resources/oleview.asp?&SD=GN&LN=EN-US&gssnb=1

2. Microsoft’s Calendar Control Documentation:

(NOTE: For properties of type Bool, -1 is TRUE and 0 is FALSE.)

Property	Type	Description
BackColor	Color	Control’s background color.
Day	Integer	The current day of the month.
DayFont	Font	The font used to display the days of the week.
DayFontColor	Color	The color used to display the days of the week.
DayLength	Integer	The format used to display the days of the week: 0 for Short (“M”), 1 for Medium (“Mon”), 2 for Long (“Monday”).
FirstDay	Integer	The day of the week to be displayed in the first column: 0 for Sunday, 1 for Monday, 2 for Tuesday, 3 for Wednesday, 4 for Thursday, 5 for Friday, 6 for Saturday

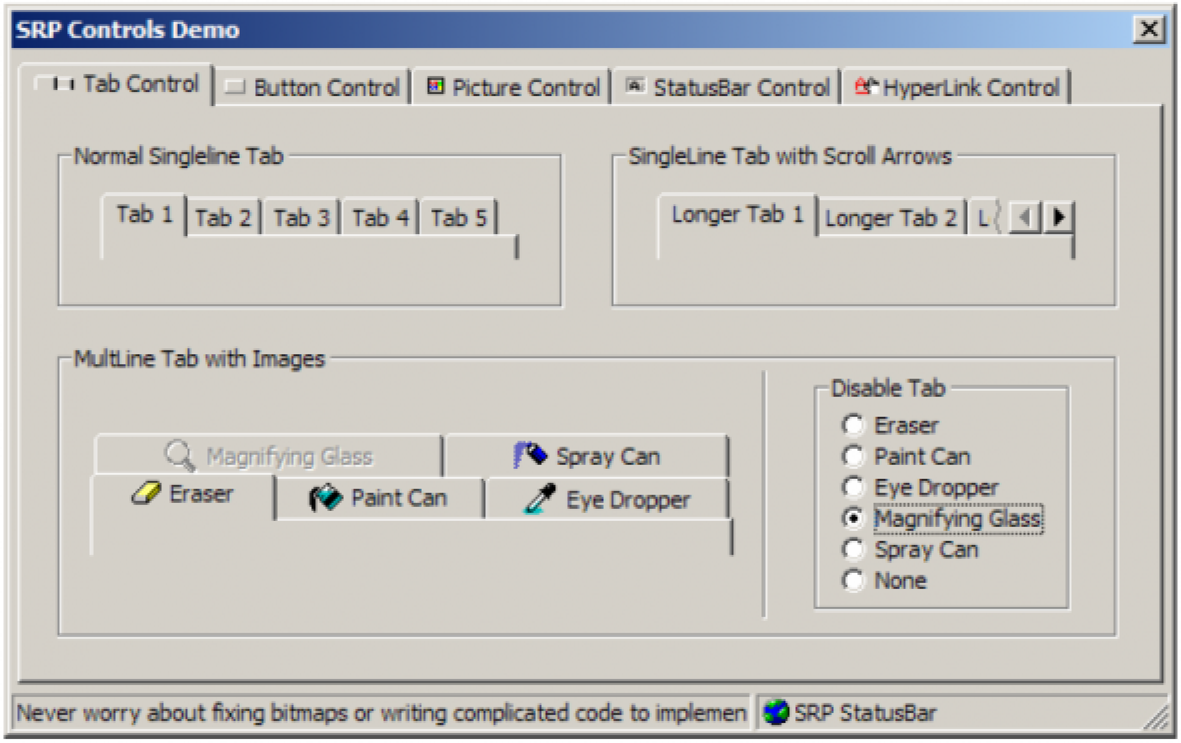
GridCellEffect	Integer	The effect used to display the grid: 0 for Flat, 1 for Raised, 2 for Sunken GridFont Font The font used to display the days of the month.
GridFontColor	Color	The color used to display the days of the month.
GridLinesColor	Color	The color used to display the lines in the grid. GridCellEffect must be 0 (Flat).
Month	Integer	The current month. A value between 1 and 12.
MonthLength	Integer	The format used to display the month: 0 for Short (“Jan”), 2 for Long (“January”).
ShowDateSelectors	Bool	Specifies whether to display drop-down boxes for the month and year.
ShowDays	Bool	Specifies whether to display the days of the week.
ShowHorizontalGrid	Bool	Specifies whether to display horizontal gridlines.
ShowTitle	Bool	Specifies whether to display the month and year above the calendar grid.
ShowVerticalGridlines	Bool	Specifies whether to display vertical gridlines.
TitleFont	Font	The font used to display the month and year above the calendar grid.
TitleFontColor	Color	The color used to display the month and year above the calendar grid.
Value	Variant	The currently selected date.
ValueIsNull	Bool	Specifies whether the value is null, i.e., no data is selected.

Method	Description
AboutBox()	Displays an informational dialog box about the Calendar Control including it’s version and copyright information.
NextDay()	Increments the control’s value by one day and refreshes the control.
NextWeek()	Increments the control’s value by one week and refreshes the control.
NextMonth()	Increments the control’s value by one month and refreshes the control.
NextYear()	Increments the control’s value by one year and refreshes the control.
PreviousDay()	Decrements the control’s value by one day and refreshes the control.
PreviousWeek()	Decrements the control’s value by one week and refreshes the control.
PreviousMonth()	Decrements the control’s value by one month and refreshes the control.
PreviousYear()	Decrements the control’s value by one year and refreshes the control.
Refresh()	Repaints the control.
Today()	Sets the control’s value to today’s data and refreshes the control.

Event	Description
AfterUpdate()	Occurs after the user selects a new date in the control and the new date is highlighted.
BeforeUpdate (Cancel)	Occurs after the user selects a new date in the control but before the new date is highlighted. If Cancel is set to TRUE (-1) then the event is canceled and the previously selected date is restored.
Click()	Occurs when the user clicks on a date in the control.
DblClick()	Occurs when the user double-clicks on a date in the control.
KeyDown (KeyCode, Shift) KeyUp(KeyCode, Shift)	Occurs when the user presses or releases a key while the control has focus. KeyCode is an integer representing the key. Shift specifies the SHIFT, CTRL, and ALT button states: 0 for None, 1 for SHIFT, 2 for CTRL, 3 for SHIFT+CTRL, 4 for ALT, 5 for SHIFT+ALT, 6 CTRL+ALT, 7 for SHIFT+CTRL+ALT.
KeyPress(KeyAscii)	Occurs when the user presses and releases a key while the control has focus. An integer that is the numeric ANSI key code.
NewMonth()	Occurs whenever the value of the control changes to display a different month.
NewYear()	Occurs whenever the value of the control changes to display a new year.

3. OpenInsight 4 Programmer's Guide, Version 4.1.3 (progref.chm), OLE event. This article contains updated information on special eventprocessing parameters that can be issued with the Send_Message() function for OLE event handling.

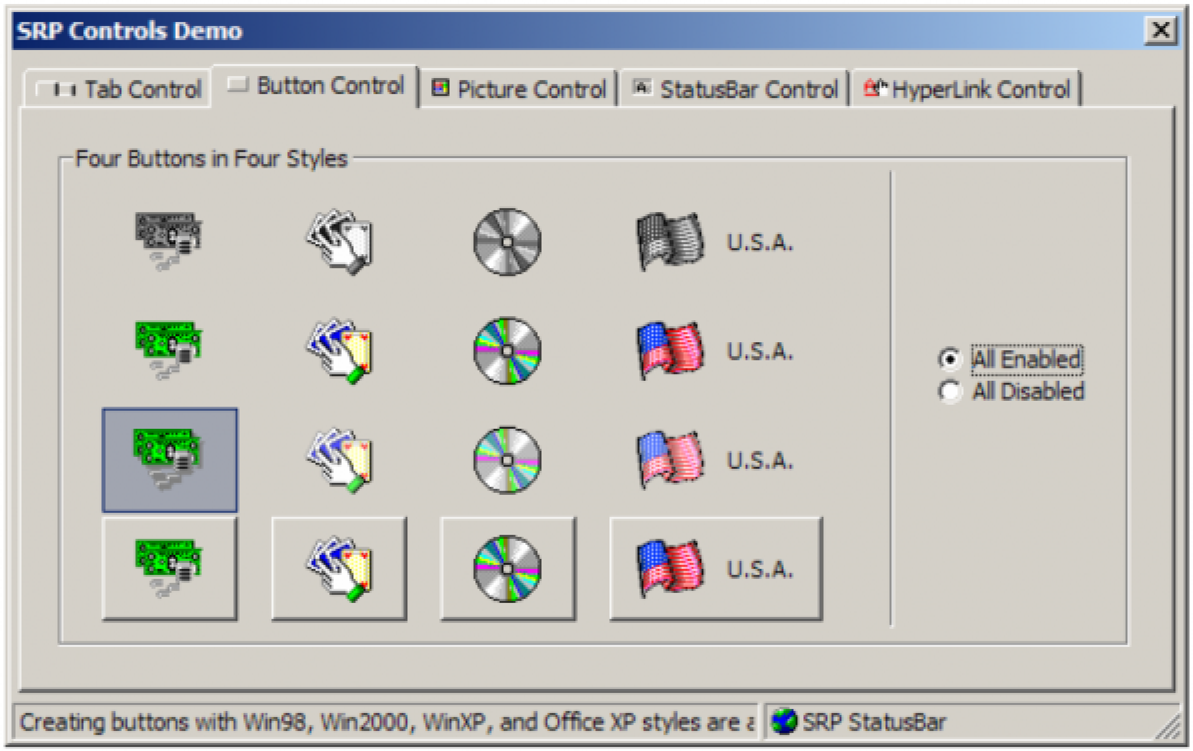
4. Revelation Software's 2002 End-of-the-Year Works CD. SRP has included evaluation versions of their own OLE controls with an OLE Demo window in the Third Party section of this CD. This includes a Tab control, Picture control, StatusBar control, Button control, and HyperLink control. Here are some screenshots of what the OLE Demo window looks like:



Disable Tab

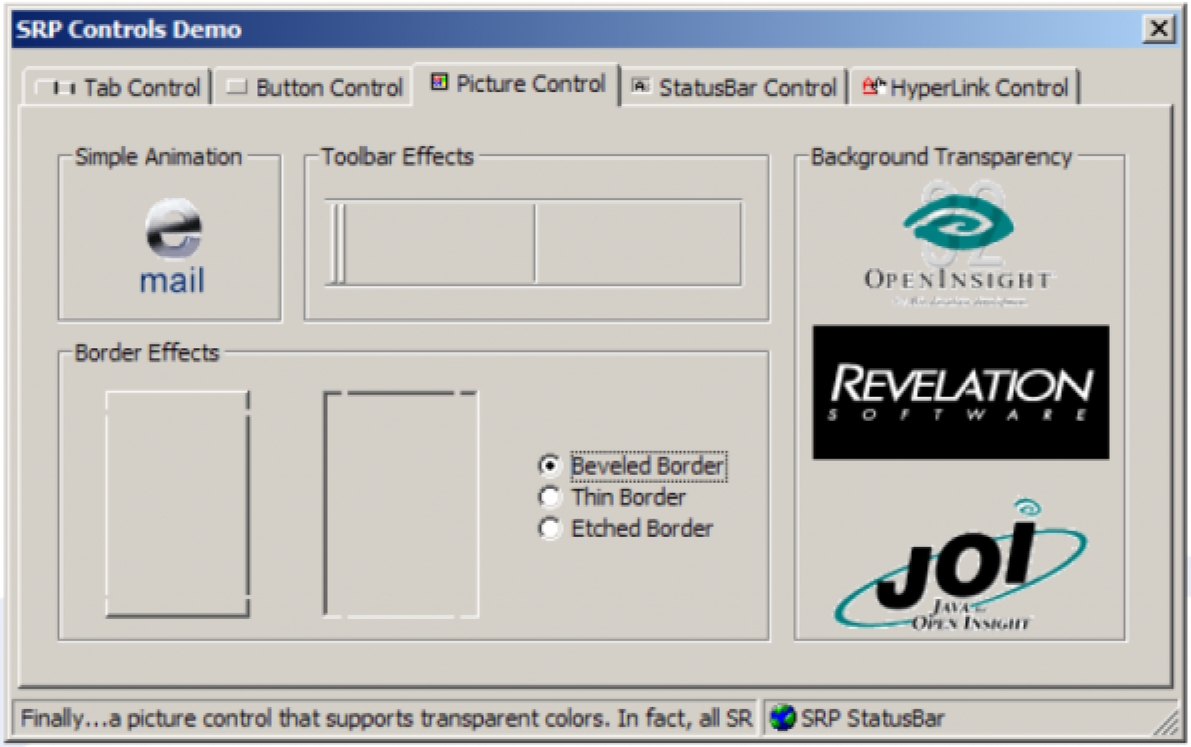
○ Eraser
○ Paint Can
○ Eye Dropper
● Magnifying Glass
○ Spray Can
○ None

Never worry about fixing bitmaps or writing complicated code to implemen



● All Enabled
○ All Disabled

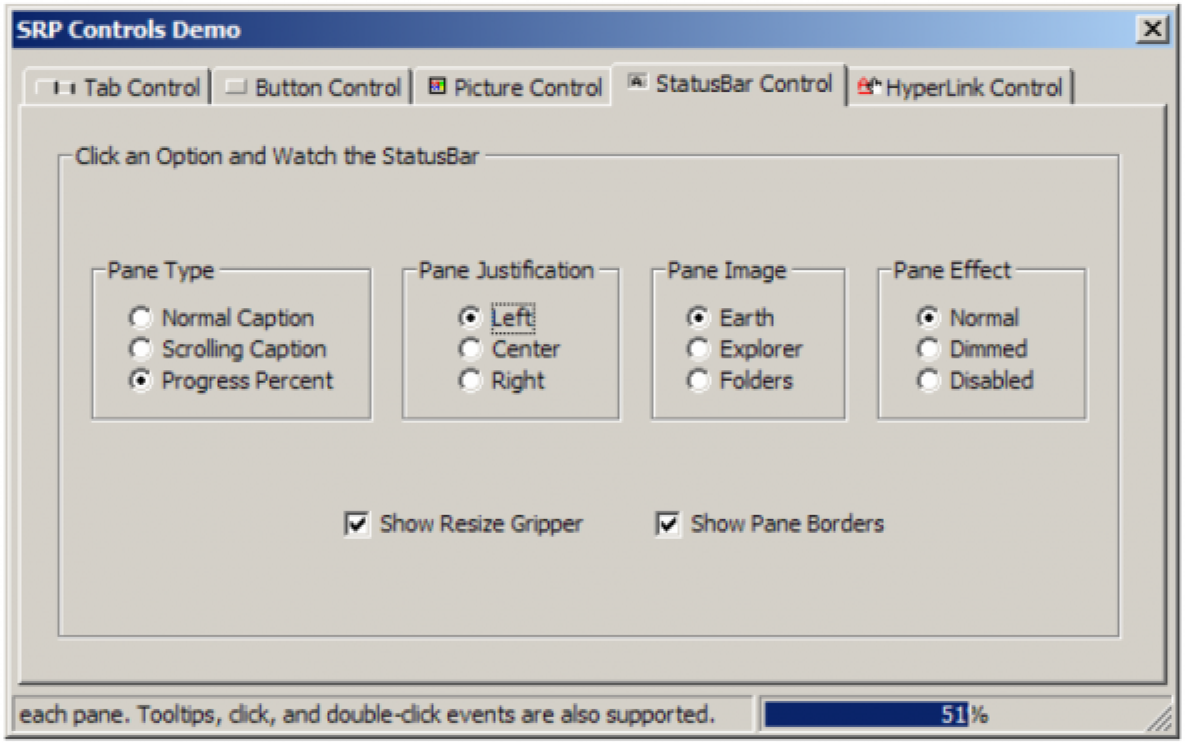
Creating buttons with Win98, Win2000, WinXP, and Office XP styles are a



SRP Controls Demo

● Beveled Border
○ Thin Border
○ Etched Border

Finally…a picture control that supports transparent colors. In fact, all SR | SRP StatusBar



SRP Controls Demo

Click an Option and Watch the StatusBar

Pane Type
○ Normal Caption
○ Scrolling Caption
● Progress Percent

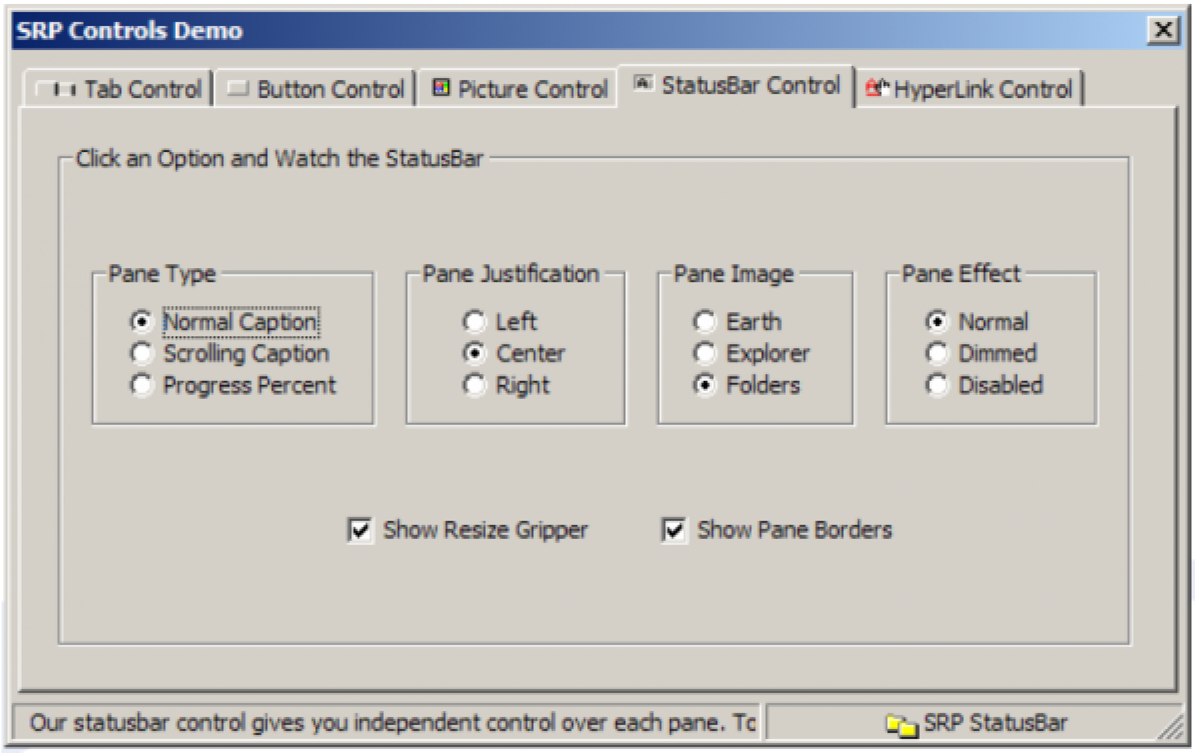
Pane Justification
● Left
○ Center
○ Right

Pane Image
● Earth
○ Explorer
○ Folders

Pane Effect
● Normal
○ Dimmed
○ Disabled

☑ Show Resize Gripper ☑ Show Pane Borders

each pane. Tooltips, click, and double-click events are also supported. | 51%



Click an Option and Watch the StatusBar

Pane Type
● Normal Caption
○ Scrolling Caption
○ Progress Percent

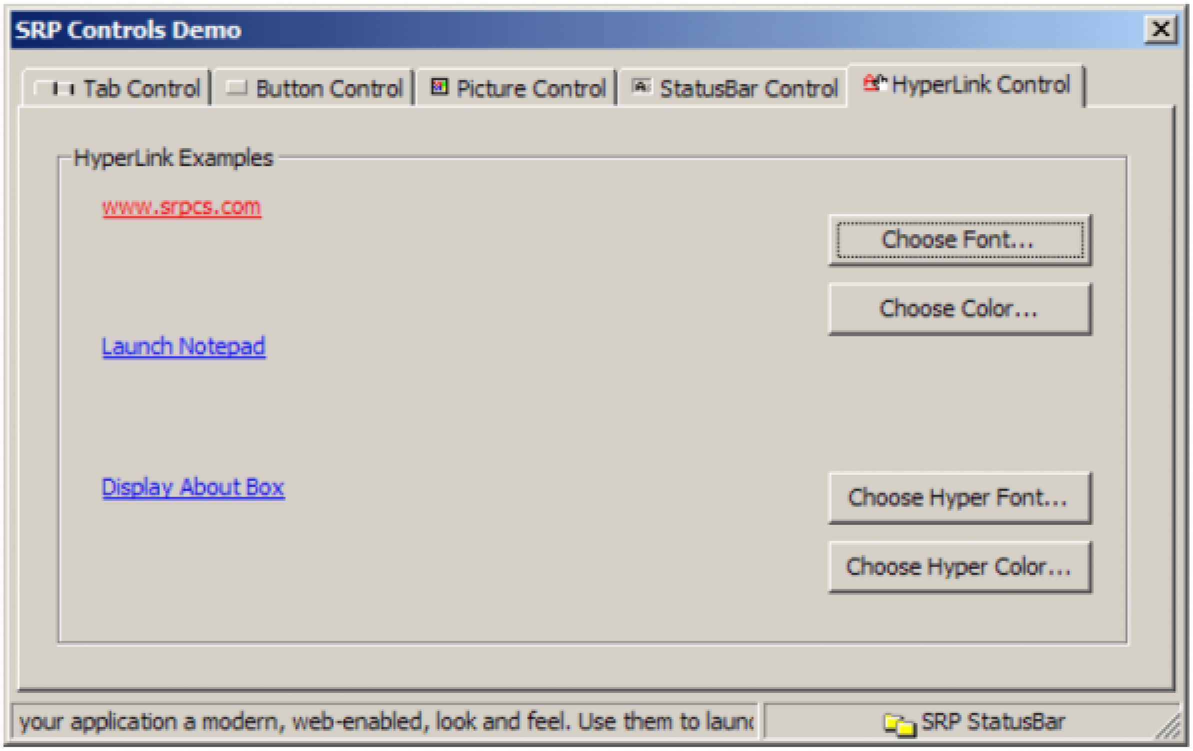
Pane Justification
○ Left
● Center
○ Right

Pane Image
○ Earth
○ Explorer
● Folders

Pane Effect
● Normal
○ Dimmed
○ Disabled

☑ Show Resize Gripper ☑ Show Pane Borders

Our statusbar control gives you independent control over each pane. Tc | SRP StatusBar



HyperLink Examples

www.srpcs.com

Launch Notepad

Display About Box

your application a modern, web-enabled, look and feel. Use them to launc | SRP StatusBar