RevDotNet Quick Start Guide

Version 1.2

REVELATION
SOFTWARE
A Division of Revelation Technologies, Inc.

Part No. 314-129

Printed in the United States of America.

Table of Contents

Section I: Using RevDotNet

RevDotNet functionality is contained in a series of APIs that OpenInsight programmers can call to create and manipulate .NET classes. These can be either visible classes (like Tree Controls, ListView Controls, etc.) or functional classes (like encryption classes, etc.). You must first establish which assemblies you wish to use, and then you may create objects from the classes in those assemblies. Once you've created an object, you can determine its methods, properties, events, etc., and invoke those methods, set and get those properties, register for those events, etc. Figure 1 shows some of the Basic+ code necessary to control the .NET ListView control.

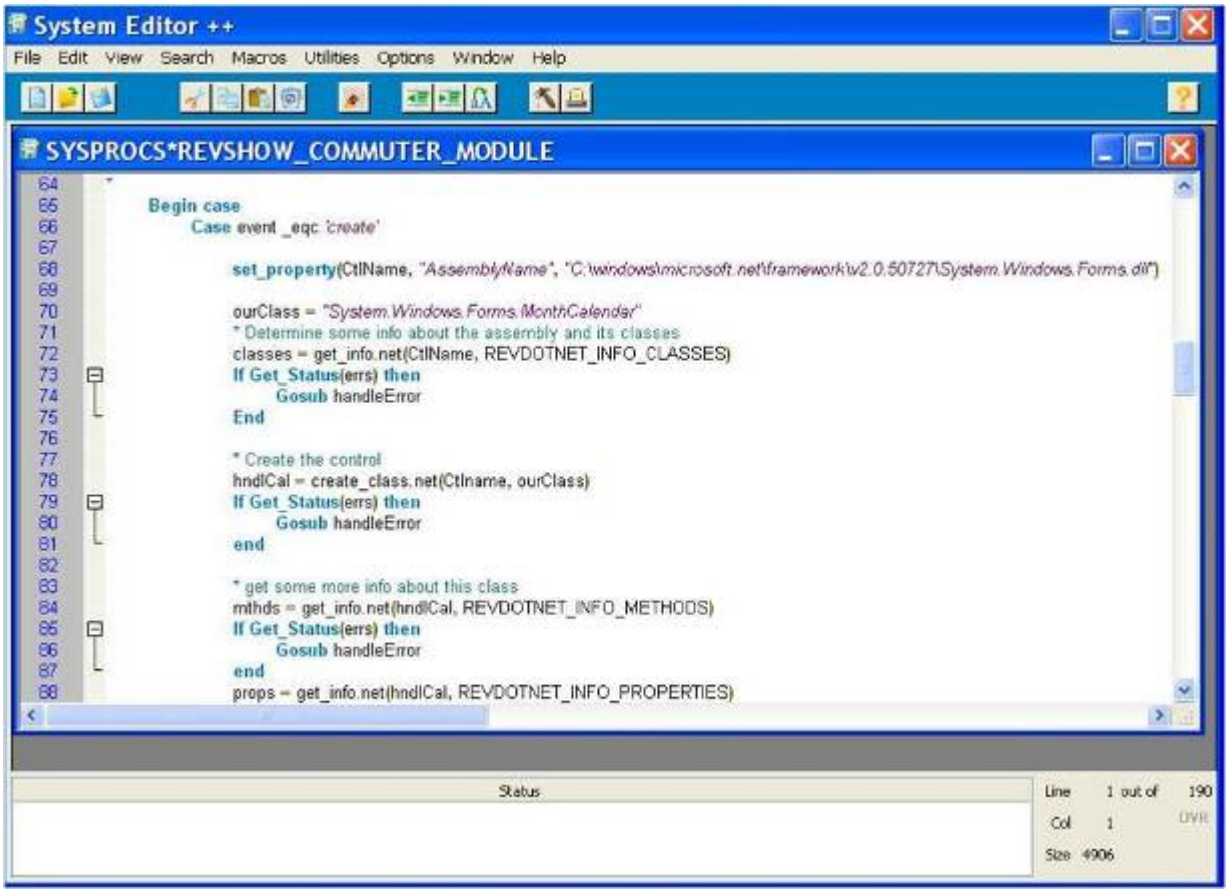


Figure 1: Communicating with .NET through the RevDotNet API in Basic+

To assist you in navigating through the capabilities of the .NET framework, the DOTNETEXPLORER tool (figure 2) can be used to examine specific classes within any assembly. Using the DOTNETEXPLORER, you can see a full list of the properties, fields, methods, events, and interfaces that a particular class contains; for visible controls, you can also manipulate the visible properties to see how changes affect the display.

The DOTNETEXPLORER may be used with .NET 2.0 controls; to investigate the properties, methods, etc. of .NET 4.0 controls, invoke the DOTNETEXPLORER4 instead.

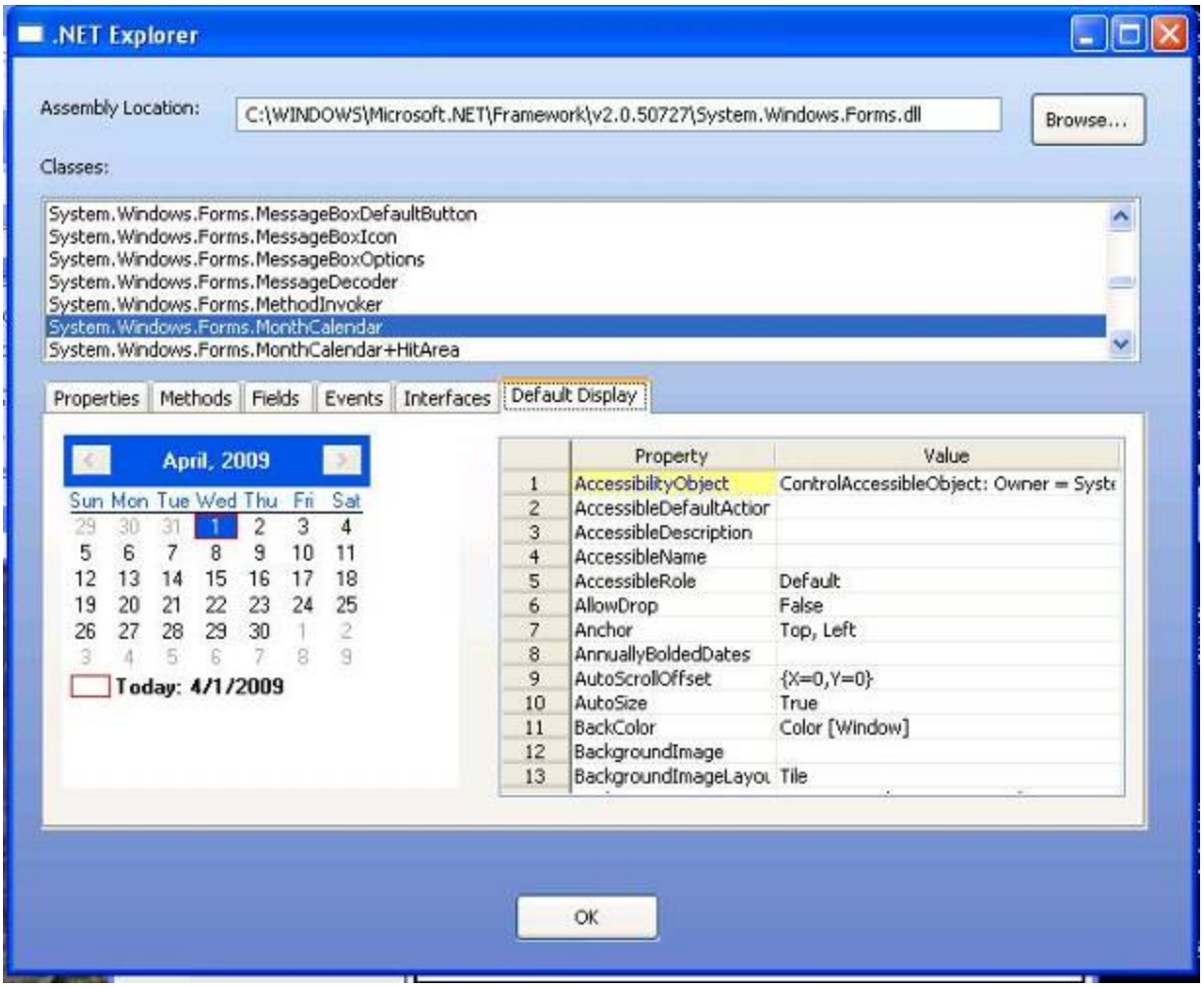


Figure 2: DotNetExplorer Interrogation/Design Tool

Using RevDotNet, you use the familiar environment of OpenInsight to manipulate all the actions of the .NET control. Note that many of these actions are fairly complex, and thus do require a significant amount of code – some of which is hidden from you when the .NET control is used, for example, in the Visual Studio environment. With RevDotNet, you get to see and control all the "behind the scenes" functionality.

Section II: RevDotNet API

Before communicating with .NET, you must first create a RevDotNet object in OpenInsight. The RevDotNet object can be created by putting a RevDotNet OCX control on a form, or it can be created "dynamically". If the .NET objects you wish to create are visible controls (for example, a calendar control), then you must use the OCX version of the RevDotNet object. If, however, you are going to be manipulating data that does not require a visible interface, then you can use either the OCX control or a dynamically-created RevDotNet object. In either case, be sure to include the REVDOTNETEQUATES insert item in your stored procedure. This insert item declares all the API functions for RevDotNet, as well as equated variables referenced in RevDotNet.

If you wish to put an actual RevDotNet control on a form, select an "OLE control" from the Form Designer toolbox, name it whatever you wish, and specify "Revelation.Invoker" as the OCX information. If you intend to communicate with a .NET 4.0 control, specify "Revelation4.Invoker" as the OCX information instead.

In either the form-based or "dynamic" RevDotNet scenario, you must "initialize" the RevDotNet environment by calling StartDotNet before using any of the RevDotNet APIs:

myHandle = StartDotNet(dotNetCtl)
or
myHandle = StartDotNet(dotNetCtl, "4.0")
when using a .NET 4.0 control.

StartDotNet returns a "handle" that will be used to communicate with the RevDotNet object. If you are using a form-based RevDotNet, then the passed-in parameter is the control ID of the RevDotNet OCX control. If you are dynamically creating a RevDotNet object, then the parameter to StartDotNet is not specified:

myHandle = StartDotNet()
or
myHandle = StartDotNet("", "4.0")
when using a .NET 4.0 control.

After creating the RevDotNet object, the AssemblyName property must be set to specify the path or paths where the .NET classes you wish to create can be found. This is done with the set_property.net function:

rslt = set_property.net(myHandle, "AssemblyName", paths)

If desired, multiple paths may be specified, @fm delimited.

To create a specific .NET object, we can call the create_class.net function. This function requires the handle to the previously-created RevDotNet object, the name of the class you wish to create, and whether this object should be visible when created, and returns a handle to the .NET object:

hndlClass = create_class.net(myHandle, ClassName, 0)

If the .NET object you wish to create requires additional parameters, you may specify the parameters and their .NET types after the "Visible?" flag:

pointHndl = create_class.net(myHandle, "System.Drawing.Point", 0, "10":@FM:"10",
"System.Int16":@FM:"System.Int16")

Multiple parameters can be specified, @fm delimited. Parameters and their types should be 'associated' (there should be the same number of types as there are values specified).

If desired (or if required in more complicated scenarios), the parameters can be "bundled together" into a single entity using create_params.net. Invoke create_params.net with a dimensioned array of parameters, a dynamic array of the parameter types, and the number of parameters:

```
DIM Params(3)
Params(1) = "Hello World"
Params(2) = "10"
Params(3)= pointHndl
pTypes = "System.String"
pTypes<2> = "System.Int16"
pTypes<3> = "RevDotNet" ;* used to represent an internal RevDotNet object
paramObject = create_params.net(3, mat Params, pTypes)
```

You can determine a variety of information about the .NET class using the get_info.net function. The get_info.net function can return details about the properties, methods, events, fields, and interfaces of any .NET class:

```
mthds = get_info.net(hndlClass, REVDOTNET_INFO_METHODS)
evts = get_info.net(hndlClass, REVDOTNET_INFO_EVENTS)
```

If you wish to capture any events that the .NET object raises, you must call the events.net API:

```
Call events.net(hndlClass, "OnForeColorChanged")
```

In addition, you must specify in your form that the RevDotNet control should respond to the OLE event, and you must handle the OLE event in your commuter module:

```
Case event _eqc "OLE" And P1 _eqc "DotNetEvent"
        caller = P2<1,1>
        event = P2<1,2>
        Call Msg(@window, "Event ":event:" has happened!":@FM:"BOK")
```

Note that the details provided by each event will vary by the type of event.

You can now proceed to set properties of the class with set_property.net:

```
rslt = set_property.net(hndlClass, "TitleForeColor", "Red")
```

You can also get properties with get_property.net:

```
currColor = get_property.net(hndlClass, "ForeColor")
```

In some cases, the get_property.net call may need to return additional .NET objects. If this is the case, specify an additional parameter to indicate that the returned value should be treated as an object (instead of as just a "normal" return value):

```
hndlNewClass = get_property.net(hndlClass, "GetSubObject", 1)
```

If you wish to return a particular value in an indexed field, you may specify the index or indices in square brackets after the property name:

```
thisSpecificValue = get_property.net(hndlClass, "ArrayValue[0]")
```

If arrays need to be passed into, or extracted out of, .NET, use the array_utility.net API. You can create an array by specifying "CREATE", either with a list of initial values:

oIntArray = array_utility.net(hndlClass, "Create", "System.Int32", "",
"1":@FM:"3":@FM:"5":@FM:"7":@FM:"9":@FM:"11") ; create a 6 element array*

or an "empty" array by specifying the size of the array:

oCharArray = array_utility.net(hndlClass, "Create", "System.Char", "1", "")*

Extract an array element with the "Get" command:

thirdPiece = array_utility.net(oIntArray, "Get", 2) ; note: 0-based counting in arrays*

Set an array element with the "Set" command:

dummy = array_utility.net(oIntArray, "Set", 3, "99") ; note: 0-based array, so this is setting the 4th element*

To invoke "methods" on the .NET object, you can use send_message.net:

rslt = send_message.net(hndlClass, "UpdateBoldedDates")

If you have parameters to specify in the send_message.net call, you must specify the parameter (and optionally the .NET type of the parameter) after the name of the method:

Rslt = send_message.net(hndlClass, "AddBoldedDate", "04/01/09", "System.DateTime")

Multiple parameters can be specified, @fm delimited; the parameter types should be "associated" with the parameter values (ie, if there are 10 values that you are passing, there should be 10 types specified as well). If you do not specify a type for a parameter value, RevDotNet will attempt to determine the appropriate type. You may also pass in a "parameter structure" created with create_params.net.

If the method you are invoking will return a new .NET object, then you must pass an additional flag at the end of the send_message.net call to indicate that you wish the return value of the function to be treated as a RevDotNet object:

Rslt = send_message.net(hndlClass, "MakeNewObject", "", "', 1)

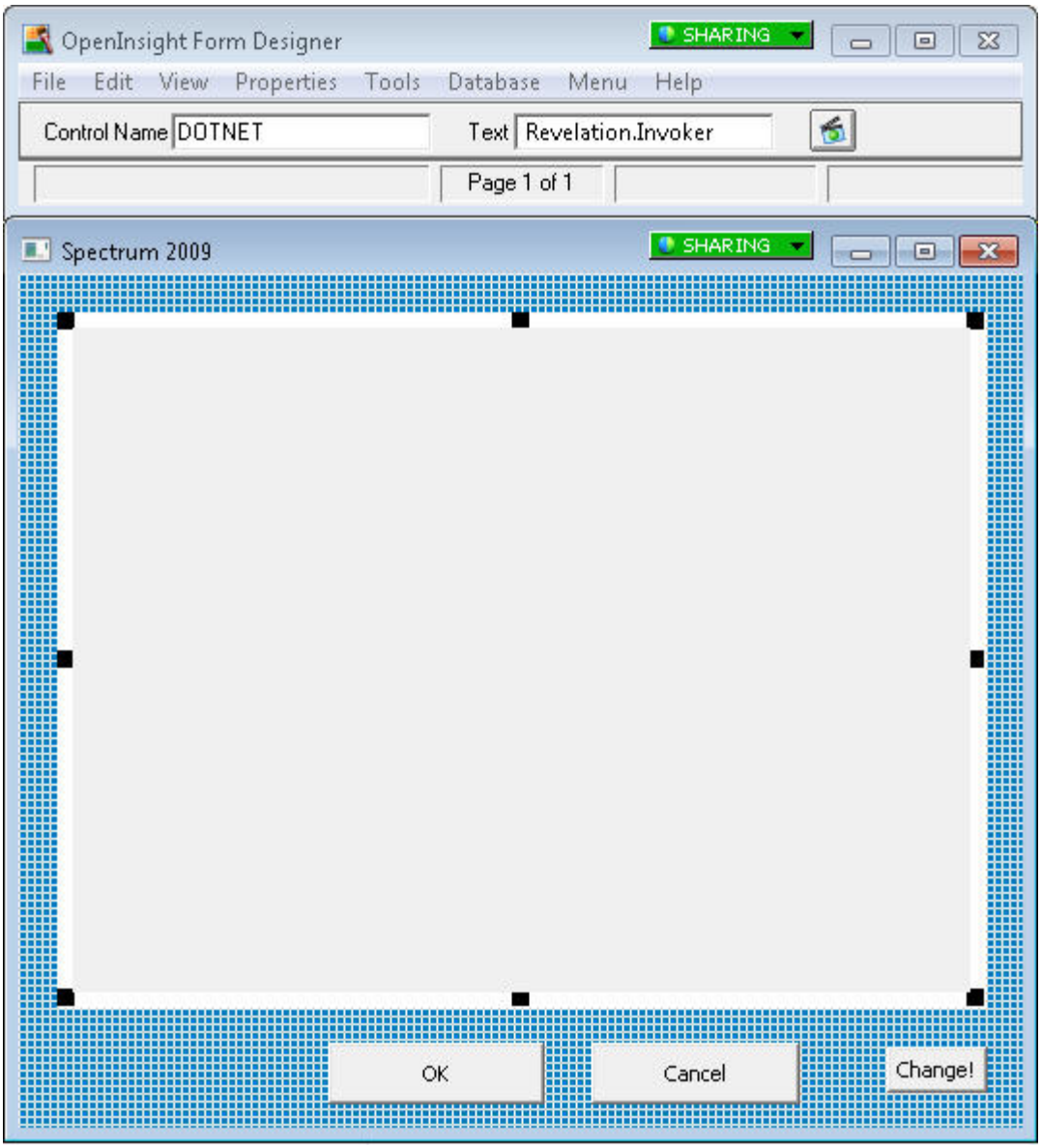
When you no longer need one of the created .NET objects, you should call the free_class.net API to remove it from memory:

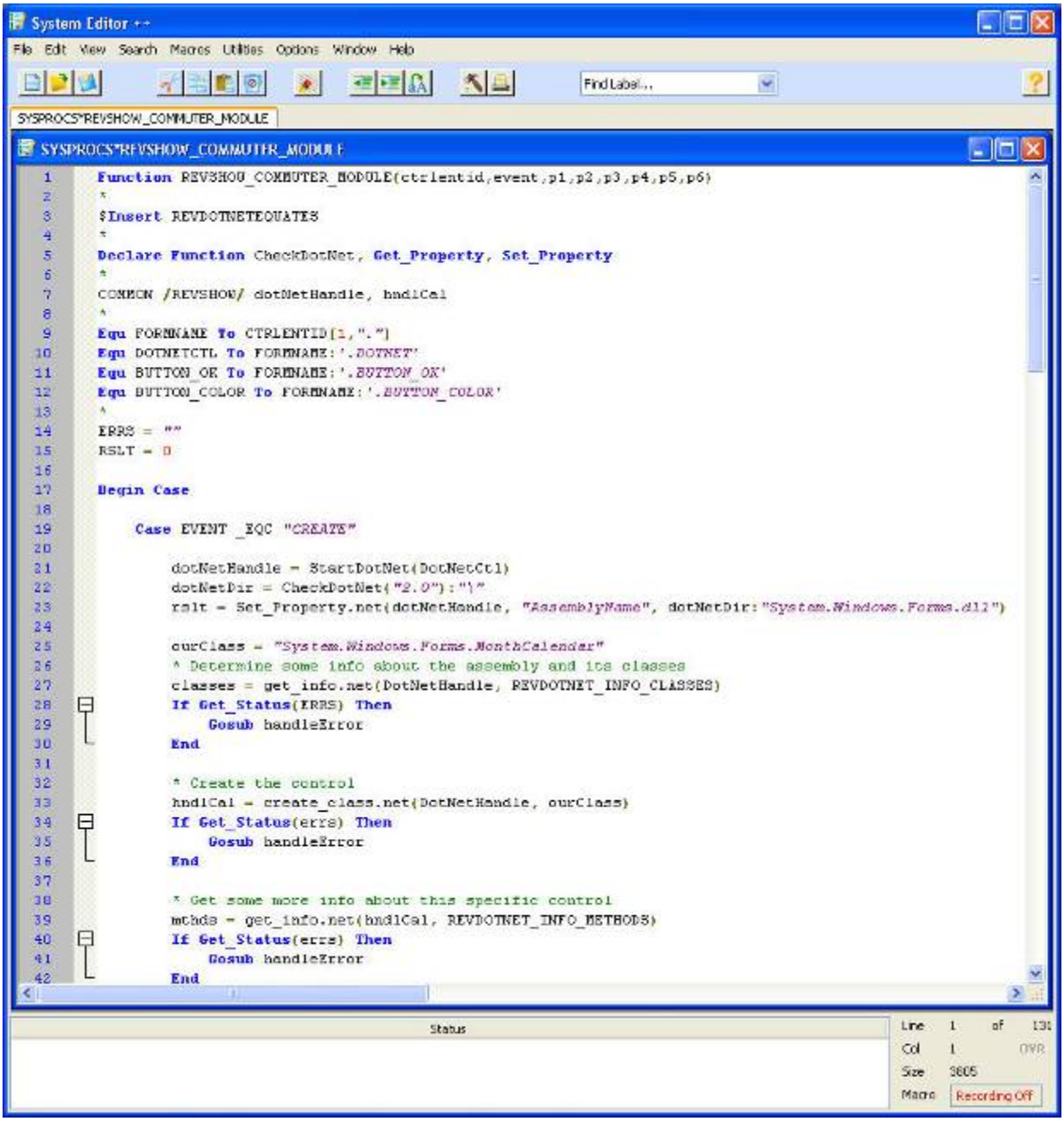
Free_class.net(hndlClass)

If free_class.net is called with an empty parameter, then *all* .NET objects will be freed:
Free_class.net()

Section III: The .NET Calendar Control

In the Form Designer, create a new form. Add the OLE control to the form, name it DOTNET, and set the TEXT property to Revelation.Invoker. Double-click on the DOTNET control, and in the "Events" for the control indicate that the OLE event should call your commuter module. Also add a button, named BUTTON_OK, and have its "click" event call your commuter module.


OpenInsight Form Designer
SHARING
File Edit View Properties Tools Database Menu Help
Control Name DOTNET
Text Revelation.Invoker
Page 1 of 1
Spectrum 2009
SHARING
OK
Cancel
Change!



In the Editor++, create your commuter module. Be sure to add the following insert:

$INSERT REVDOTNETEQUATES

Declare a named common to hold the "handles" to the RevDotNet objects, some helper functions, and some equates to make our code easier to understand:

Declare Function CheckDotNet, Get_Property, Set_Property
COMMON /REVSHOW/ dotNetHandle, hndlCal
Equ FORMNAME To CTRLENTID[1,"."]
Equ DOTNETCTL To FORMNAME:'.DOTNET'
Equ BUTTON_OK To FORMNAME:'.BUTTON_OK'

On the “create” event of the form, initialize the RevDotNet interface:

```
Case EVENT _EQC "CREATE"
dotNetHandle = StartDotNet(DotNetCtl)
dotNetDir = CheckDotNet("2.0"):"\"
rslt = Set_Property.net(dotNetHandle, "AssemblyName", dotNetDir:"System.Windows.Forms.dll")
```

Note the use of the CheckDotNet utility routine; this function returns the directory where the .NET assemblies are stored (for the specified version of .NET).

Now that the path to the .NET assemblies has been set, we can create an instance of the calendar object:

```
ourClass = "System.Windows.Forms.MonthCalendar"
* Create the control
hndlCal = create_class.net(DotNetHandle, ourClass)
If Get_Status(errs) Then
        Gosub handleError
End
```

Note the use of the Get_Status function to determine if any errors occurred during the RevDotNet API call. Also note that no “Visible?” flag is passed; create_class.net will assume that .NET objects created using the RevDotNet OCX are visible, while those created using the “dynamic” object are not.

Once the object is created, you can determine additional details about it using the get_info.net API call, you can indicate which events you wish to handle in your commuter module, and you can start to manipulate the object with set_property.net, get_property.net, and send_message.net:

```
* Get some more info about this specific control
mthds = get_info.net(hndlCal, REVDOTNET_INFO_METHODS)
If Get_Status(errs) Then
        Gosub handleError
        End
mthd = get_info.net(hndlCal, REVDOTNET_INFO_METHODS, "AddBoldedDate")
If Get_Status(errs) Then
        Gosub handleError
End
props = get_info.net(hndlCal, REVDOTNET_INFO_PROPERTIES)
If Get_Status(errs) Then
        Gosub handleError
End
prop = get_info.net(hndlCal, REVDOTNET_INFO_PROPERTIES, "TitleForeColor")
If Get_Status(errs) Then
        Gosub handleError
End
events = get_info.net(hndlCal, REVDOTNET_INFO_EVENTS)
If Get_Status(errs) Then
        Gosub handleError
End
* Set up the events we care about
events.net(hndlCal, 'ForeColorChanged')
events.net(hndlCal, 'DateSelected')
* Show we can change a property
rslt = set_property.net(hndlCal, "TitleForeColor", "Red")
rslt = Set_property.net(hndlCal, "SelectionStart", "03/18/2009")
rslt = set_property.net(hndlCal, "SelectionEnd", "03/26/2009")
```

```
rslt = set_property.net(hndlCal, "ShowWeekNumbers", "True")
* And invoke a method
rslt = send_message.net(hndlCal, "AddBoldedDate", "2009-04-22", "System.DateTime")
rslt = send_message.net(hndlCal, "AddBoldedDate", "04/26/2009", "System.DateTime")
rslt = send_message.net(hndlCal, "UpdateBoldedDates")
```

Note how the get_info.net API can return information on *all* the properties, methods, events, etc., or on a specific property, method, event, etc.

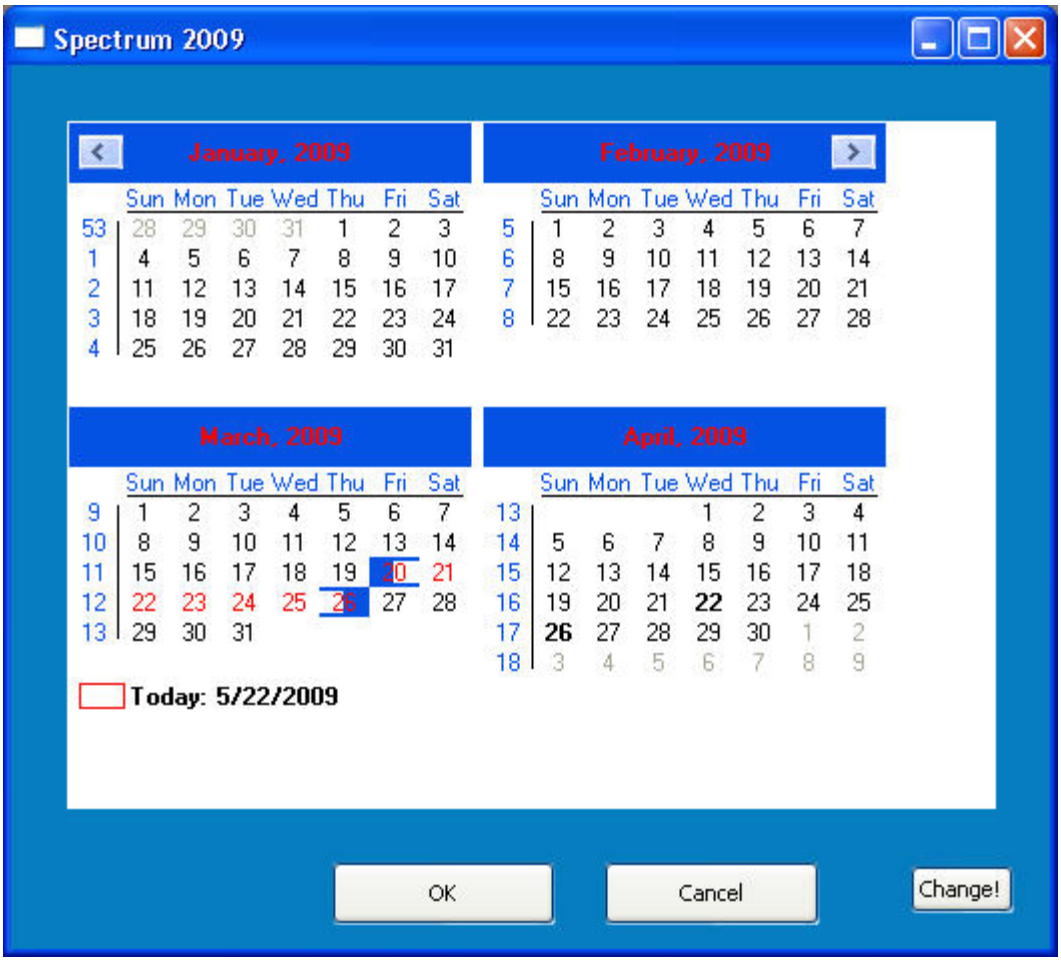
When an event on the form has been triggered, your commuter module can handle it:

```
Case event _eqc "OLE" And P1 _eqc "DotNetEvent"
        caller = P2<1,1>
        event = P2<1,2>
        If EVENT _EQC "ONDATESELECTED" Then
                STDT = P2<1,4,1>
                EDDT = P2<1,4,2>
                If STDT = EDDT Then
                        OUTSTR = STDT
                End Else
                        OUTSTR = STDT:'-':EDDT
                End
                Call Msg(@window, “You selected “: OUTSTR:@FM:”BOK”)
        End ELSE
                Call Msg(@window, "Event ":event:" has happened!":@FM:"BOK")
        END
```

Note that different types of events will return different information; this information can be found in the passed parameter.

When the user clicks the OK button, capture the currently selected dates and close the form:

```
Case CtrlEntId _eqc BUTTON_OK
selStart = get_property.net(hndlCal, "SelectionStart")
selEnd = get_property.net(hndlCal, "SelectionEnd")
DATE1 = Iconv(Field(selStart," ",1), "D")
DATE2 = Iconv(Field(selEnd," ",1), "D")
OutDate = Oconv(DATE1,'D4/')
If DATE1 <> DATE2 Then
        OutDate := " - ":Oconv(Date2,'D4/')
END
Call Msg(@WINDOW, "Selected date(s): ":OutDate:@FM:"BOK")
Call Send_event(FormName, "CLOSE")
```



The form and commuter module are available for your reference; please look at the REVSHOW form and the REVSHOW_COMMUTER_MODULE source for further information.

Revelation Software, Inc
99 Kinderkamack Road Ste 109
Westwood, NJ 07675
U.S.A
Toll Free: 800-262-4747
Phone: 201-594-1422
Fax: 201-722-9815
www.revelation.com

Revelation Software Ltd.
Boundary House
Boston Road
London, W7 2QE
U.K.
Phone: +44 0 208 912 1000
Fax: +44 0 208 912 1001
info@revsoft.co.uk

BrightIdeas New Zealand
44 Cockle Bay Rd, Howick
Auckland, 2014
New Zealand
Phone: +64 9 534 9134
info@revelation.asia

Revelation Software is a division of Revelation Technologies, Inc.

Part No. 314-129